Aplicación de *Deep Learning* al aprendizaje de modelos en robótica cognitiva

Deep Learning application to model learning in cognitive robotics

Ariel Rodríguez-Jiménez[1], Esteban Arias-Méndez[2], Francisco Bellas-Bouza[3], Jose Becerra-Permuy[4]

Rodríguez-Jiménez, A., Arias-Méndez, E., Bellas-Bouza, F., Becerra-Permuy, J. Aplicación de *Deep Learning* al aprendizaje de modelos en robótica cognitiva. Tecnología en marcha. Edición especial Movilidad Estudiantil 7. Abril, 2020. Pág. 92-104.

https://doi.org/10.18845/tm.v33i6.5171

1 Ingeniero en Computación. SimMachines, Costa Rica. Correo electrónico: arieli13.10@gmail.com

2 Profesor de Ingeniería en Computación. Escuela Ingeniería en Computación, Instituto Tecnológico de Costa Rica. Costa Rica. Correo electrónico: esteban.arias@tec.ac.cr

3 Profesor titular de universidad. Escuela Politécnica Superior, Departamento de Computación, Universidad de la Coruña, España. Correo electrónico: francisco.bellas@udc.es

4 Profesor titular de universidad. Escuela Politécnica Superior, Departamento de Computación, Universidad de la Coruña, España.Correo electrónico: jose.antonio.becerra.permuy@udc.es

Palabras clave

Red neuronal artificial; entrenamiento con batch; entrenamiento con mini-batch; aprendizaje online; aprendizaje profundo; robótica cognitiva; aprendizaje automático; optimizadores estocásticos; robot Baxter.

Resumen

El tipo de entrenamiento utilizado para una red neuronal artificial va a depender de factores como: disponibilidad de datos, tiempo de entrenamiento, recursos de hardware disponibles, entre otros. Los entrenamientos pueden ser offline u online. En el presente artículo se experimentan entrenamientos online sobre un robot cuya principal característica es que utiliza un mecanismo cognitivo darwinista para su supervivencia.

El aprendizaje del robot se produce en tiempo real. Este tiene redes neuronales artificiales profundas para predecir acciones a realizar, se entrena con la menor cantidad de espacio de almacenamiento y en el menor tiempo posible sin sacrificar confianza de la red neuronal artificial profunda.

Los entrenamientos experimentados son: Online Deep Learning, Online Deep Learning con memoria y Online Mini-Batch Deep Learning con memoria.

Keywords

Artificial neural network; batch training; mini-batch training; online learning; deep learning; cognitive robotics; machine learning; stochastic optimizer; Baxter robot.

Abstract

The kind of training used for an artificial neural network will depend on factors such as: available data, training time, hardware resources, etc. The trainings can be online and offline. In the current article we experimented with online trainings on a robot whose main characteristic is the usage of a Cognitive Darwinist Mechanism to survive.

The robot learns in real-time. It has deep artificial neural networks to predict actions, it's trained using the least amount of storage and the training time has to be as fast as possible; keeping high confidence in the artificial neural network.

The experimental trainings are: Online Deep Learning, Online Deep Learning with memory and Online Mini-Batch Deep Learning with memory.

Introducción

La robótica cognitiva [1] es un campo que estudia la creación de arquitecturas para robots, permitiéndoles aprender progresivamente mientras interactúan con el mundo real con dispositivos como cámaras, micrófonos, parlantes, etc.

La idea principal de este artículo es exponer un mecanismo de Machine Learning capaz de aprender modelos matemáticos, prediciendo futuras acciones relacionadas a distintas entradas (de información) basadas en acciones previamente realizadas. Las redes neuronales artificiales pueden modelar problemas de clasificación, regresión, entre otros; sim embargo, en este artículo nos enfocamos en el segundo [2].

Mientras el robot está interactuando con el ambiente, se obtiene nueva información por aprender, entonces se puede decir que no se dispone de un dataset completo para entrenar el modelo (al inicio), este es entrenado con un solo dato a la vez. Esto significa que los métodos de entrenamiento comunes como batch [3] no se pueden utilizar para resolver este problema; no obstante, Online Learning [9] es el apropiado porque con optimizadores estocásticos somos capaces de aprender el modelo descrito anteriormente.

Es importante escoger un método de entrenamiento para la red neuronal artificial, pero también es muy importante tener una arquitectura robusta para hacer el aprendizaje más sencillo y efectivo. Deep Learning [4] puede hacer modelos más complejos, aprender distintas dimensiones de datos y además mejorar la confianza de la red.

Robótica cognitiva

La robótica cognitiva [1], se centra en la creación de arquitecturas de desarrollo para robots que les permita realizar un aprendizaje autónomo progresivo a partir de su propia interacción con el mundo real. Esta aproximación contrasta con la robótica clásica en la que se pre-programa el robot con respuestas específicas a tareas predefinidas.

Descripción del problema

El robot Baxter [7] es un modelo de mundo [6] que aprende en tiempo real a predecir acciones que en un futuro le pueden generar una pérdida o una ganancia. Llámese pérdida a las acciones que anteriormente se ejecutaron y produjeron ganancia, pero se volvieron a ejecutar en un contexto similar y no se obtuvo recompensa; por ejemplo: hay un objetivo el cual es ingresar una pelota en una caja. El robot tiene la bola en la mano y la caja al frente entonces la acción que realiza es meter la bola en la caja, esto le produce una ganancia. En algún otro momento el objetivo cambia y ahora es agarrar la caja. Supongamos que el robot se encuentra con la bola en la mano y la caja al frente (ambiente similar al anterior), el robot decide meter la bola en la caja y termina sus acciones. Esto no produce ninguna recompensa pues el objetivo cambió, así que se añade una pérdida a los datos pues para ese ambiente, el objetivo es distinto y las acciones también. Las acciones que se pueden ejecutar por el robot son las siguientes:

- Agarrar objeto
- Pedir favor (Ejemplo: Pedir que acerquen una caja, a través de parlantes)
- Cambiar objeto de mano
- Poner objeto en una caja
- Arrastrar objeto
- Agarrar con ambas manos
- Tirar objeto

El objetivo de los experimentos es poder aprender basándose en los resultados que le produjeron las acciones que ha ejecutado anteriormente. Es decir, saber que si su objetivo era agarrar la bola y lo que hizo fue golpearla, aprender que esa acción para este objetivo, no le puede traer ningún beneficio entonces tiene una probabilidad baja de ejecución, mientras que agarrar la bola tiene una probabilidad alta pues anteriormente le dio un buen resultado. Cada acción que se quiere predecir va a depender completamente del contexto actual en el que se encuentre el robot. Porque el ambiente es muy poco probable que sea idéntico a los anteriores,

es por esto que las predicciones son probabilidades ([0, 1] probabilidad de ejecutar o no una acción). La idea es que por cada acción (llámese acción a ejecuciones desde un movimiento que pueda hacer el robot como mover una mano, hasta moverse a un lado y agarrar un objeto) haya un mecanismo de aprendizaje automático que pueda decidir qué tan probable es que sea buena para lograr un objetivo. Podemos decir que un objetivo puede componerse de una o más acciones, es por esto que las acciones futuras dependen de las anteriores, y se debe de aprender de estas, es decir: olvidar lo menos posible. La adopción del robot al ambiente se realiza utilizando una arquitectura MDB [6]. Dentro de la memoria de largo plazo existen modelos de aprendizaje automático cuya función es predecir si ejecutar o no cierta acción (como mover alguna parte, o puede ser incluso un grupo de acciones); estos son llamados pnodes. Los pnodes reciben información del contexto actual (ambiente en el que se encuentra el robot mediante sensores como cámaras, micrófonos…) y predicen la probabilidad de que la acción que les corresponde sea buena o mala para el objetivo propuesto. Ejemplo: Para agarrar una bola que se encuentra al lado derecho del robot, es necesario activar los pnodes que tienen como objetivo predecir si: mover el hombro derecho, mover la mano derecha, agarrar un objeto; van a generar alguna recompensa. Para poder realizar este trabajo, cada pnode recibe la siguiente información:

- ball_in_right_hand: Booleano que indica si el robot tiene la bola agarrada con la mano derecha, en metros.
- ball_dist: Distancia que hay entre el robot y la bola, en metros.
- box_size: Tamaño de la caja, en metros.
- ball_in_left_hand: Booleano que indica si el robot tiene la bola agarrada con la mano izquierda.
- box_ang: Ángulo del robot a la caja, en radianes entre [-π, π].
- ball_ang: Ángulo del robot a la bola, en radianes entre [-π, π].
- box_dist: Distancia que hay entre el robot y la caja, en metros.
- ball_size: Tamaño de la bola, en metros.

Como se puede observar, los objetivos del robot para estos pnodes serían meter la bola en una caja, mover una caja, agarrar una bola, etc. Se trabajó con 21 pnodes que cada uno predice la ejecución de una acción de las mencionadas anteriormente.

Metodología

Es importante saber que el aprendizaje del robot es en tiempo real, los datos ingresan 1 por 1 de manera secuencial al modelo de aprendizaje automático, y se necesita algún mecanismo que ayude a aprender el modelo matemático del problema. Para poder retornar buenas acciones, lo que se hace es: hacer una predicción, y posteriormente recibir una respuesta indicando si la acción realizada estuvo buena o mala (Este feedback de la acción lo indica un sensor que tiene el robot, del cual desconozco su funcionamiento). De esta manera, entrenar el modelo de predicciones (cada pnode que participó en la acción) para ajustarlo y aprender de la acción realizada. El tiempo que le toma al robot realizar una acción puede ser bastante largo (1 min, por ejemplo), por ello lo que se hizo fue simular la ejecución de acciones por parte del robot con un programa de simulación, y almacenar los datos obtenidos para poder entrenar un modelo de aprendizaje automático. De esta manera se obtuvo un dataset por cada pnode para poder entrenar y probar los modelos. (Cada dataset con datos entre 250 y 1000 registros aproximadamente).

Para lograr el aprendizaje del robot se utilizaba un algoritmo evolutivo llamado Neat [11] como el modelo de apredizaje automático (para cada pnode). El algoritmo lo que hace es buscar una topología de red neuronal óptima para la predicción de los datos. El objetivo era encontrar un método de Deep Learning [4] que mejore el resultado de las predicciones, olvide muy poco para no perder información y que cada vez que entrene le tome poco tiempo; pues es un trabajo que realiza en tiempo real (como se mencionó anteriormente).

Solución implementada

Existen dos tipos de entrenamiento para redes neuronales: online [10] y offline. La diferencia entre estos dos radica principalmente en la manera en que la red neuronal artificial es actualizada y la forma como ingresan los datos de entrenamiento. Los datos en un entrenamiento online[9] normalmente se obtienen 1 a 1 de manera secuencial, es decir no se tiene el dataset completo pues al principio no se dispone de datos para su entrenamiento. Es por esto que se entrena un dato a la vez, y se ejecuta el algoritmo de optimización con cada dato (Stachastic Grandient Descent [2], Adam [8], Adagrad [5], etc), además el learning rate [12] es bajo (~0.001, depende mucho del problema) pues la red prácticamente nunca deja de entrenarse (en problemas de tiempo real, como el que se está tratando). Para el entrenamiento de estas redes se utilizan optimizadores estocásticos, es decir, ejecutan el algoritmo de optimización cada vez que se recibe un nuevo dato. La idea es que la red aprenda de los nuevos sin olvidar los anteriores. Por otro lado, el entrenamiento offline (los típicos con batch [3] y mini-batch [3]) realizan un entrenamiento con el dataset completo, y una vez la red converge podría estar lista para ser utilizada.

Está claro que el tipo de entrenamiento que necesitamos es online, porque la red es alimentada con los datos que le brinde el robot cada vez que ejecute una acción y esta tenga un resultado ya sea positivo o negativo. Como la idea es que el robot nunca deje de aprender, se podría decir que se dispone de un dataset infinito, aunque al principio es nulo (las decisiones que tome al comenzar su ciclo de vida no serán optimas, podría decirse que serán aleatorias).

Para solucionar el problema se decidió utilizar 3 tipos de entrenamiento online [9] debido a que se al ser de tiempo real es lo que se recomienda por la comunidad científica. Los 3 entrenamientos están basados en Online Deep Learning [10]:

- Online Deep Learning [10].
- Online Deep Learning [10] con memoria.
- Online Mini-Batch Deep Learning con memoria.

Arquitectura de la red neuronal artificial

Para lograr un resultado satisfactorio es necesario construir una red neuronal artificial capaz de aprender un modelo matemático que sea similar al del problema tratado. Se experimentó con distintos tamaños de red (entre 2 y 5 capas escondidas) con diferentes números de neuronas en cada capa. Las capas de entrada y salida tenían 8 y 1 neuronas respectivamente.

Las funciones de activación de las capas intermedias fueron RELU (Rectilinear Unit), mientras que la de output fue Sigmoid pues el resultado debía de ser una probabilidad (comprendida entre [0, 1]).

De los dataset, se utilizó aproximadamente el 70% de los registros como conjunto de entrenamiento y el 30% como conjunto de prueba.

Por otra parte, para comparar las topologías de red, se realizaron entrenamientos de tipo Mini-Batch [3] con el dataset completo, (cada mini-batch tenía 5 registros) fueron entrenamientos asistidos donde se terminaba cuando la red alcanzaba la convergencia.

El learning rate [12] utilizado varió entre 0.1, 0.01 y 0.001; con el objetivo de tener distintas referencias y ver el cambio tanto en tiempo de entrenamiento como en aprendizaje.

La calidad de la red se midió basándose en el error final del conjunto de test, cuanto menor fuera éste mejor era la red. Este error era calculado con el Error Cuadrático Medio, cuya fórmula es:

$$\sqrt{\frac{1}{n} * \sum_{i=1}^{n} \left(\overline{x}_i - x_i \right)}$$

Para la actualización de los pesos de la red se utilizó el optimizador ADAM [8].

El cuadro 1 muestra los resultados de las 5 mejores topologías de red obtenidas (todas ellas con un learning rate [12] de 0.01).

Cuadro 1. Resultados de las 5 mejores topologías de red obtenidas.

Número de neuronas en capas escondidas	Iteraciones hasta alcanzar convergencia	Raíz del error cuadrático medio del conjunto de prueba	STDDEV promedio del conjunto de prueba
128, 64, 64, 32	2096	0.003658	0.055114
16, 16, 16, 8	3038	0.004749	0.057674
128, 64	3320	0.005097	0.056115
128, 64, 64, 32, 16	1547	0.005486	0.061715
32, 32, 16	2113	0.005517	0.067645

La red escogida para realizar los entrenamientos online fue la que tiene 4 capas escondidas con 128, 64, 64 y 32 neuronas respectivamente, con funciones de activación RELU en cada capa intermedia y Sigmoid en la capa de salida.

Entrenamientos

Online Deep Learning

Cada vez que la red recibe un nuevo registro para entrenar, ésta lo realiza y luego del cálculo del error y actualización de sus pesos el registro es desechado. Este entrenamiento tiene la ventaja de que no gasta memoria almacenando un dataset. Sin embargo, el aprendizaje es muy costoso. Es muy sencillo caer en overfitting o underfitting si el learning rate [12] no es el adecuado. Por otro lado, es un entrenamiento que requiere bastante uso de los recursos de hardware pues ejecuta el algoritmo de optimización con cada nuevo registro.

Código ilustrativo:

```
while new_data:
    input, label = split(new_data)
    for step in range(train_steps):
        prediction = model.predict(input)
        cost = cost_function(prediction, label)
        model.update(cost)
```

Online Deep Learning con memoria

La idea de este algoritmo es mantener una memoria donde se almacenan N registros ingresados previamente. Cada vez que la red entrena con un dato, este es almacenado en la memoria. Si ésta se llena, se procede a eliminar un registro para ingresar el nuevo. Los métodos de selección del registro a eliminar pueden ser variados, desde un algoritmo tipo cola hasta selección por menor cantidad de vistas, etc.

Cada vez que ingresa un nuevo dato se entrena de manera online con la memoria completa. Ejecutando N iteraciones.

Código ilustrativo:

```
mem = Queue(SIZE)
While new_data:
    mem.enqueue(new_data)
    for step in range(train_steps):
        for inputs, labels in mem:
            predictions = model.predict(input)
            cost = cost_function(prediction, labels)
            model.update(cost)
```

Online Mini-Batch Deep Learning con memoria

A diferencia con los entrenamientos online comunes, éste actualiza los pesos de la red cada N registros procesados. El entrenamiento online [9] normal (como el mencionado anteriormente) ejecuta el algoritmo de optimización cada vez que recibe un registro, mientras que con Mini-Batch [3] ejecuta el algoritmo de optimización cada N registros. Se calcula el error medio de los N registros procesados y con ese dato se actualizan los pesos. La idea es dividir la memoria en M batches, cada batch de tamaño N.

Código ilustrativo:

```
mem = Queue(SIZE)
While new_data:
    mem.enqueue(new_data)
    mini_batches = mini_batch_queue(mem, mini_batch_size)
    for step in range(train_steps):
        for inputs, labels in mini_batches:
            predictions = model.predict(input)
            cost = cost_function(prediction, labels)
            model.update(cost)
```

Resultados

Para los entrenamientos se utilizó un dataset que tenía 665 datos, 500 se designaron a entrenamiento y los otros 165 para validación.

Los tres entrenamientos fueron ejecutados variando learning rate [12] (0.1, 0.01, 0.001), número de iteraciones y tamaño de memoria (los que la tienen). A continuación, se muestra el mejor resultado de cada uno.

En las gráficas de dispersión, los puntos naranjas son el objetivo y los puntos azules son los que la red neuronal predijo.

Online Deep Learning

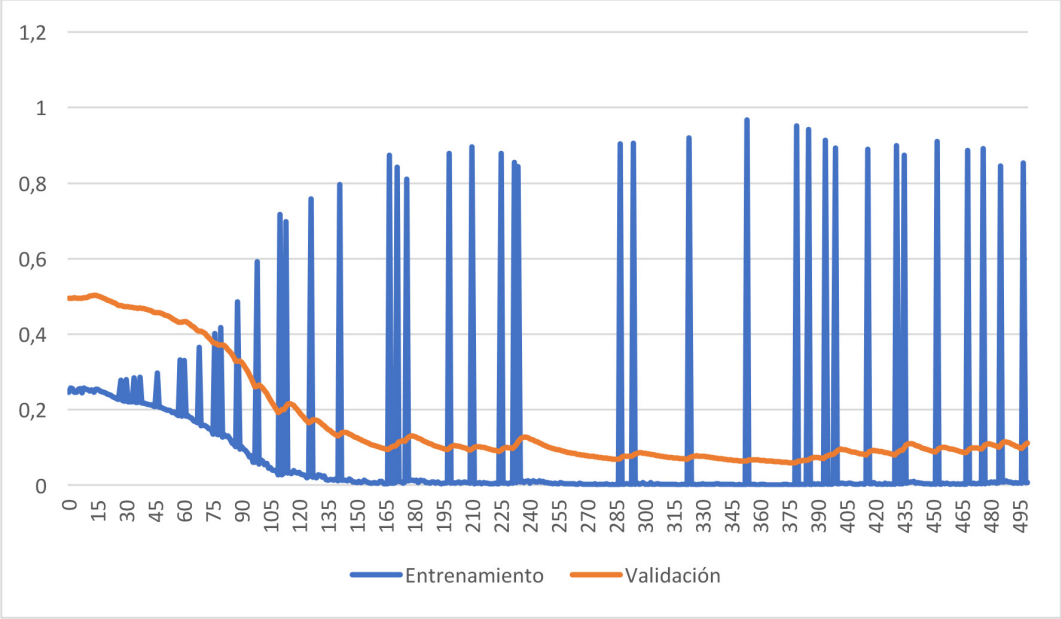

Figura 1. Gráfico lineal. En el eje X se muestra el número de iteración del entrenamiento Online Deep Learning [10], en el eje Y se muestra el error cuadrático medio sobre el conjunto de datos.

Por cada iteración se realizaron 5 pasos de entrenamiento con el mismo registro. Como se puede observar la gráfica de entrenamiento no llega a converger de la manera esperada. Hay muchos puntos donde el error de entrenamiento aumenta de manera exagerada, se puede observar que la red no aprendió el modelo. Incluso en la gráfica de validación nunca se estabiliza el error.

Las predicciones finales de este entrenamiento se observan en la siguiente gráfica de dispersión.

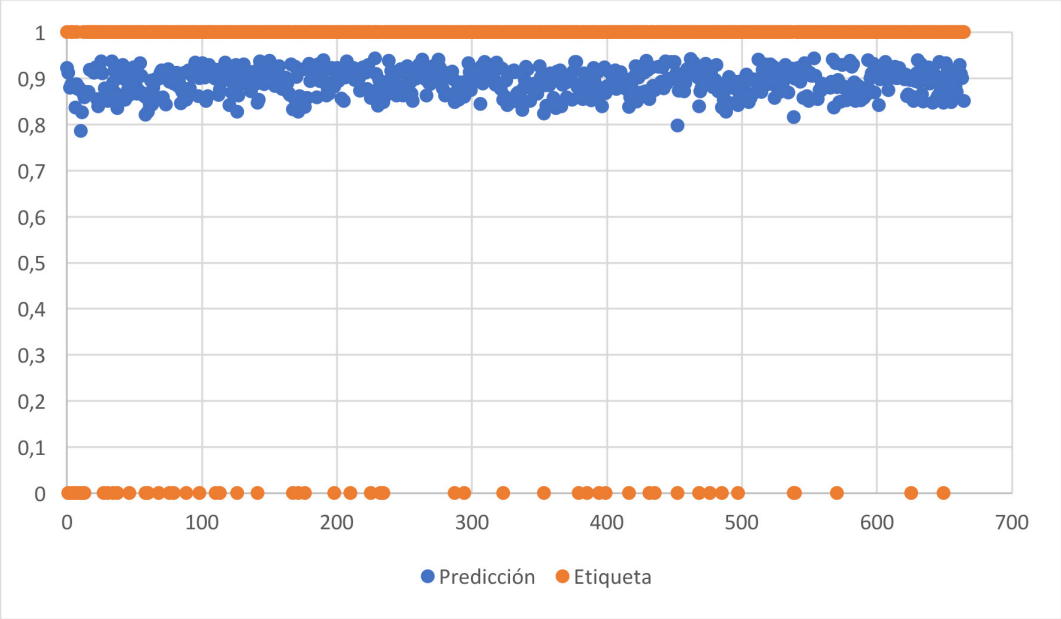

Figura 2. Gráfico de dispersión. Los puntos anaranjados son las etiquetas mientras que los puntos azules son las predicciones de la red neuronal.

Las predicciones finales de prueba, como se muestran en la figura 2, muestran que no se logró aprender el modelo, por lo tanto, la red no es confiable.

Online Deep Learning con memoria

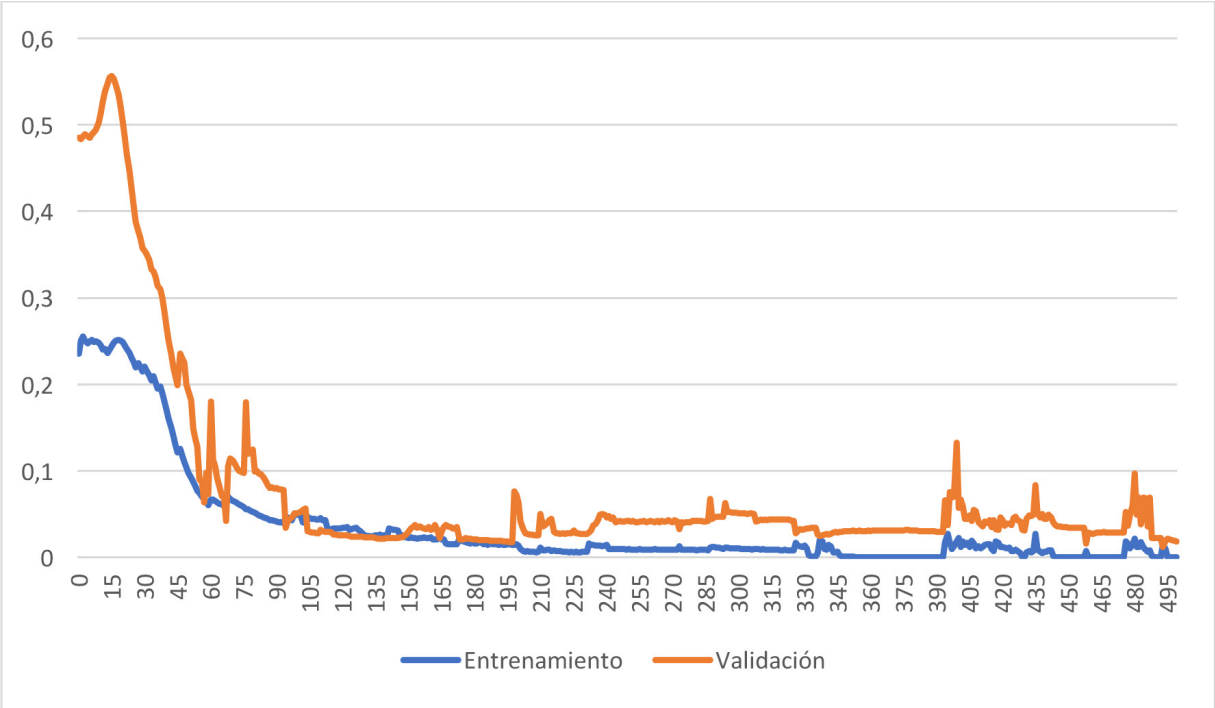

Figura 3. Gráfico lineal. En el eje X se muestra el número de iteración del entrenamiento Online Deep Learning [10] con memoria, en el eje Y se muestra el error cuadrático medio sobre el conjunto de datos.

Para este entrenamiento se utilizó una memoria tipo cola de tamaño 100. Además, el número de pasos de entrenamiento fueron 25.

El error de entrenamiento tiene menos varianza que el de la figura1. Además, hay menos oscilaciones cada vez que recibe un nuevo registro.

Las predicciones finales de este entrenamiento se observan en la siguiente gráfica de dispersión.

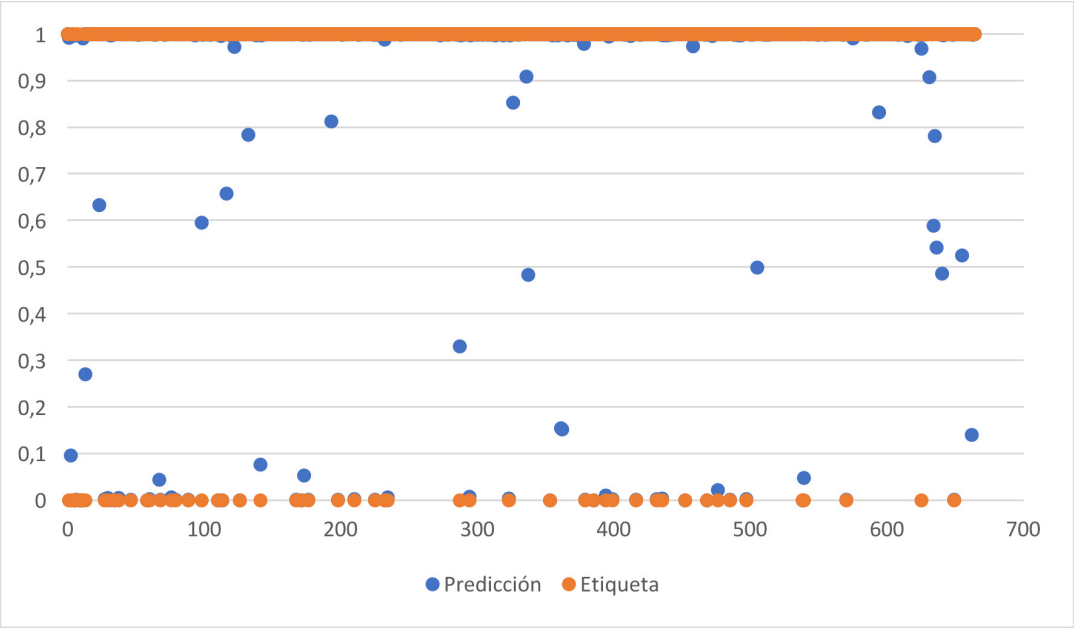

Figura 4. Gráfico de dispersión. Los puntos anaranjados son las etiquetas mientras que los puntos azules son las predicciones de la red neuronal.

En la figura 4 el aprendizaje de la red fue mejor, se nota en las predicciones que hay una mayor confianza porque son más cercanas a sus respectivas etiquetas.

Online Mini-Batch Deep Learning con memoria

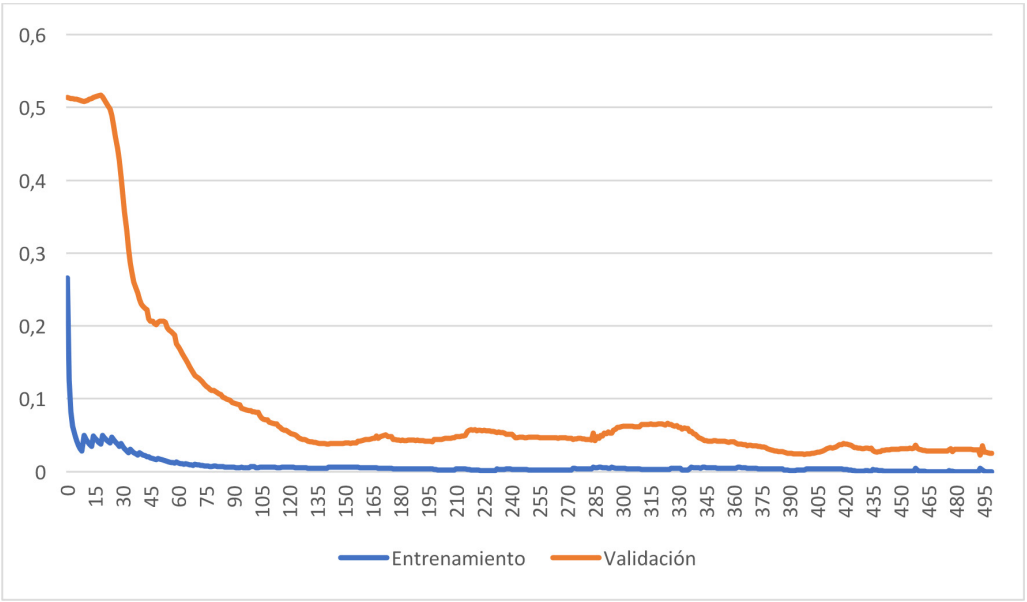

Figura 5. Gráfico lineal. En el eje X se muestra el número de iteración del entrenamiento Online Mini-Batch Deep Learning con memoria, en el eje Y se muestra el error cuadrático medio sobre el conjunto de datos.

Este entrenamiento fue el mejor de los tres porque ambas gráficas presentan menos latencia y alcanzan una mejor convergencia.

Las predicciones finales de este entrenamiento se observan en la siguiente gráfica de dispersión (figura 6).

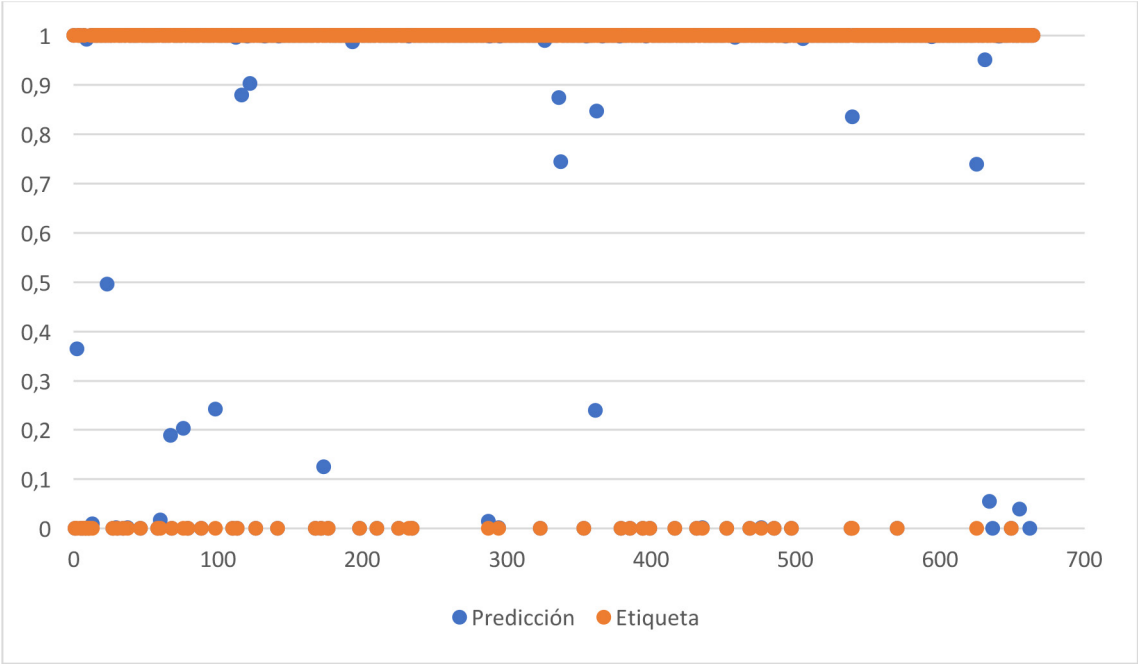

Figura 6. Gráfico de dispersión. Los puntos anaranjados son las etiquetas mientras que los puntos azules son las predicciones de la red neuronal.

Finalmente, en la figura 6 se presenta la gráfica de dispersión de la mejor red neuronal obtenida de todos los experimentos. Se aprecia que la distancia entre etiqueta y predicción es menor y además la mayoría es acertada.

Conclusión

El aprendizaje en tiempo real de una red neuronal no es viable realizarlo mediante técnicas de entrenamiento comunes. Es necesario utilizar entrenamientos que permitan aprender muchos datos con la menor cantidad de iteraciones (sobre estos) posible, para de esta manera evitar problemas como el overfitting y underfitting.

Para que un modelo de mundo cognitivo aprenda de esta manera, es necesario tomar en cuenta factores como uso de la memoria y el rendimiento; de esta manera garantizamos un aprendizaje poco costoso computacionalmente hablando, y eficiente. Es necesario tomar en cuenta estas características porque el modelo de mundo tiene otras funciones que estar ejecutando mientras se realiza el aprendizaje, no se puede dedicar solo a ello.

El aprendizaje con el que se obtuvo mejores resultados (Online Mini-Batch Learning con memoria) toma en cuenta todos los factores mencionados anteriormente. Este entrenamiento funciona en modelos que dispongan de poca memoria de almacenamiento y una arquitectura de hardware no muy potente.

La arquitectura de la red se puede mejorar experimentado con más capas escondidas, agregando o eliminando más o menos neuronas, utilizando otras funciones de activación, etc.

Por otra parte, el método de escoger cuál registro eliminar de la memoria también; por ejemplo, utilizando algoritmos que eliminen registros similares, algoritmos que traten mantener en la memoria la mayor dispersión de datos posible, algoritmos que mantienen los registros más vistos, entre otros.

El objetivo del artículo fue exponer el nuevo método de entrenamiento experimentado (Online Mini-Batch Deep Learning con memoria) y los resultados fueron satisfactorios pues mejoró el aprendizaje en comparación con los otros dos métodos, mejorando la convergencia de la red tanto en entrenamiento como en validación.

Referencias

[1] Arrabales, R. (2007). Robótica Cognitiva (1era edición) [Online]. Disponible: http://www.conscious-robots.com/es/2007/08/21/robotica-cognitiva/

[2] Bottou, U. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent (1era edición) [Online]. Disponible: https://leon.bottou.org/publications/pdf/compstat-2010.pdf

[3] Brownlee, J. (2017). A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size (1era edición) [Online]. Disponible: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[4] Crowford, C. (2016). An introduction to deep learning (1era edición) [Online]. Disponible: https://blog.algorithmia.com/introduction-to-deep-learning/

[5] Duchi, S., Hazan, E. y Singer, Y. (2011). Adaptive Subgradient Method for Online Learning and Stochastic Optimization (1era edición) [Online]. Disponible: http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[6] F. Bellas, “MDB mecanismo cognitivo darwinista para agentes autónomos”, Disertación doctoral, departamento de computación, Universidade da Coruña, A Coruña, España, 2003.

[7] Guizzo, B. y Ackerman, E. (2018, setiembre 18). How Rethink Robotics Built its New Baxter Robot Worker (1era edición) [Online]. Available: https://spectrum.ieee.org/robotics/industrial-robots/rethink-robotics-baxter-robot-factory-worker

[8] Kingma, D., y Lei J. (2015). ADAM: A method for stochastic optimization (1era edición) [Online]. Disponible: https://arxiv.org/pdf/1412.6980.pdf

[9] Poggio, T., Voinea, S., y Rosasco, L. (2018). Online learning, stability and stochastic gradient descent (1era edición) [Online]. Disponible: https://arxiv.org/pdf/1105.4701.pdf

[10] Sahoo, D., Pham, Q., Lu J., y C.H S. (2017). Online Deep Learning: Learning Deep Neural Networks on the Fly (1era edición) [Online]. Disponible: https://arxiv.org/pdf/1711.03705.pdf

[11] Stanley, K. y Miikkulainen, R. (2002). Evolving Neural Networks Through Augmenting Topologies (1era edición) [Online]. Disponible: http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

[12] Zulkifli, H. (2018). Understanding Learning Rates and How It Improves Performance in Deep Learning (1era edición) [Online]. Disponible: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10