Optimización y control del flujo de materiales en procesos de producción flexibles utilizando aprendizaje profundo

Carolina Saavedra Sueldo*, Ivo Perez Colo*, Mariano De Paula *, Sebastian A. Villar*and Gerardo G. Acosta*
*INTELYMEC, Centro de Investigaciones en Física e Ingeniería del Centro,
UNICEN – CICpBA – CONICET, B7400JWI, Olavarría, Argentina
Emails: {carolina.saavedra; ivo.perez; mariano.depaula;svillar ggacosta}@fio.unicen.edu.ar

***Abstract*—Industry 4.0, currently on the rise, demands increasing flexibility and adaptation of production systems to changing products demands and external factors. The adaptation of the production systems implies frequent and often abrupt changes in the configurations of the shop floors and consequently the movement of materials must be re-planned. Material handling is significant in terms of operative costs and times and it does not add value to the end products. It is desired to optimize the performance of the system based on the degree of movements, buffer usage and waiting times, such that the combination of these minimizes the impact on the process costs. Machine learning algorithms in combination with powerful computational simulators can be mutually leveraged to give rise to solve these kinds of real-world problems, typical of smart factories. In this work, for the optimization approach, we develop a closed-loop decision-making system with a deep reinforcement learning algorithm based on a discrete-event simulation model for material handling. In addition, our proposed approach uses the communication architecture *Simulai*, which allows interfacing a computational discrete-event simulator and the proposed deep learning-based decision-making algorithm. The functionality of our proposal is evidenced through the obtained results and an optimal solution for the problem stated is reached, proving that an intelligent agent can collaborate in making multiple decisions for smart factories.**

***Index Terms*—Industry 4.0, Autonomous Decision System, Deep Reinforcement Learning, Optimization, Material Handling.**

I. Introducción

En la actualidad, la Inteligencia Artificial (IA) y el aprendizaje automático se utilizan en diversos ámbitos y problemas de la realidad, buscando nuevas soluciones a inconvenientes o maximizando y optimizando resultados.

La mejora de los sistemas productivos alineada a la Cuarta Revolución Industrial o Industria 4.0, supone la combinación del acelerado desarrollo tecnológico, técnicas y métodos existentes y los sistemas industriales actuales para dar lugar a sistemas inteligentes como se describen en [1]. Se trata de un cambio radical aprovechando la revolución digital en software, robótica, internet de las cosas, dispositivos conectados, sistemas ciberfísicos e inteligencia artificial conformando nuevas industrias automatizadas e innovadoras que optimizan la utilización de recursos mediante las más diversas aplicaciones [2]. Todas estas herramientas forman pilares para el proceso de toma de decisiones dentro de las organizaciones y su combinación genera sistemas de mayor robustez, permitiendo reducir los riesgos que implican los cambios, del contexto interno como externo. Por lo tanto, los sistemas de producción modernos requieren mayor flexibilidad de forma tal de ganar capacidad, adaptabilidad, agilidad y seguridad en los desafíos que impone el entorno [3].

En esta nueva filosofía de producción personalizada, se acentúa cada vez más el uso de recursos y servicios rentados dentro de los sistemas productivos para conseguir una alta flexibilidad y capacidad de reconfiguración [4]. Todo sistema de producción flexible se centra en la sinergia del conjunto de procesos que lo componen. De esta forma surge el concepto de manufactura sincrónica [5], el cual impulsa el control y la coordinación de los recursos para mejorar el desempeño global del sistema. Esto implica considerar elementos como movimientos, lugares, tiempos, espacios y cantidades, de los cuales se ocupa el manejo de materiales.

Los modelos de simulación representan procesos y situaciones complejas de la realidad. De esta forma, los mismos permiten conceptualizar un proceso productivo y su gestión en términos de los eventos que puedan suceder y cuyo impacto afecta a distintos actores del sistema. Las ventajas de su utilización demuestran la rapidez al momento de analizar problemas específicos y la capacidad visual que otorgan para la interpretación de los flujos productivos [6]. Sin embargo, el dinamismo que deben lograr las organizaciones para tomar decisiones de bajo riesgo implica la obtención, procesamiento e interpretación de grandes cantidades de datos que requieren herramientas modernas sofisticadas.

Por lo tanto, con el auge de la Industria 4.0, los problemas relacionados a manufactura flexible, manejo de materiales, calidad, logística, entre otros, también comienzan a ser resueltos con IA. Incluso vemos el aprendizaje profundo aplicado a sectores como el mercado de capitales o la planificación de rutas. En [7] exploran una combinación de estrategias de aprendizaje profundo para predecir diferentes mercados futuros y tomar decisiones con mayor certidumbre, al igual que en [8], [9], [10] y [11] donde usan *Deep Reinforcement Learning* para trazar estrategias de inversión que les permitan obtener mejores rendimientos. En [12] optimizan el algoritmo *Deep Q-Network* (DQN) para planificar la ruta óptima de un barco de acuerdo con reglas de navegación reales, mejorando la seguridad y la toma de decisiones. En

ISBN: 978-987-88-2891-6

[13] también planifican rutas con IA, pero en este caso para robots AGVs en un almacén. Otro campo de aplicación es la optimización de parámetros, como en [14], donde utilizan DQN en una red blockchain generada para el procesamiento de grandes cantidades de datos provenientes de redes IOT. Además, hay varios ejemplos de aplicación de DQN en problemas de planificación de la producción, como en [15], [16] y [17].

Con el objetivo de lograr la consolidación entre las diversas tecnologías, métodos y técnicas que nos propone esta Cuarta Revolución Industrial, surge la necesidad de incorporar nuevos elementos que aseguren el flujo de datos. De esta forma nace, en el núcleo de trabajo, una herramienta citada anteriormente, *Simulai* [18]. Sintéticamente, se trata de una librería modular desarrollada e implementada como un paquete en lenguaje Python. La misma permite integrar, por ejemplo, un modelo previamente diseñado en un software de simulación de eventos discretos con un módulo de toma de decisiones, el cual puede estar basado en técnicas de IA como es el caso de algún algoritmo de aprendizaje por refuerzos.

En este trabajo, haciendo uso de *Simulai*, se implementa el algoritmo DQN para encontrar la mejor solución a un problema de manejo de materiales en piso de planta, donde se busca optimizar el desempeño del sistema basándose en la cantidad de movimientos, uso de *buffers* y tiempos de espera. Se comprueba que este algoritmo funciona correctamente para resolverlo, logrando minimizar los costos asociados a estas tres variables del sistema.

A continuación, se presenta la metodología utilizada, seguida del caso de estudio implementado, los resultados obtenidos y las conclusiones finales del trabajo.

II. Metodología

A. Aprendizaje por Refuerzo

En el Aprendizaje por Refuerzo (AR) o *Reinforcement Learning*, un agente trata de aprender qué hacer en una determinada situación basándose en estímulos (recompensas) obtenidos por el efecto de sus decisiones. En otras palabras, el agente artificial aprende cómo asignar acciones a situaciones, para maximizar una señal de recompensa numérica con el fin de resolver una tarea de toma de decisiones. En el AR, no se le ”dice” de antemano al agente (el tomador de decisiones) qué acciones tomar en determinadas situaciones, sino que debe descubrir aquellas que generan la mayor recompensa esperada. En los casos más interesantes y desafiantes, las acciones pueden afectar no sólo la transición inmediata, sino que también pueden tener un efecto retardado en la evolución del sistema [19].

En un problema de aprendizaje por refuerzos, el agente interactúa con el entorno durante una secuencia de pasos de tiempo discretos, tomando decisiones y obteniendo recompensas. Formalmente, este proceso se conoce como proceso de decisión de Markov, en el cual en cada paso de tiempo *t*, el agente recibe alguna representación del estado del ambiente s_t y en función de dicho estado selecciona una acción a_t siguiendo una política de actuación $\pi(s_t)$. Un paso de tiempo después, y como consecuencia de la acción tomada, recibe una recompensa numérica r_{t+1} y evoluciona a un nuevo estado s_{t+1}. Lo que se busca en el entrenamiento del sistema es maximizar esta recompensa [19]. Normalmente, los algoritmos de AR cuando trabajan en grandes espacios de estados y acciones emplean aproximadores funcionales, como por ejemplo redes neuronales profundas, para aproximar su política de actuación $\pi(\cdot)$ y las funciones de valor empleadas en dichos algoritmos [20].

B. Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNA) son sistemas de procesamiento, hardware o software, que copian esquemáticamente la estructura neuronal del cerebro para tratar de reproducir sus capacidades. Por lo tanto, son capaces de aprender de la experiencia a partir de señales o datos provenientes del exterior. Los tres conceptos claves de los sistemas nerviosos que se pretenden emular en las redes artificiales son: paralelismo de cálculo, memoria distribuida y adaptabilidad al entorno [21].

La estructura de un sistema neuronal artificial es similar, jerárquicamente, a uno biológico. El elemento esencial es la neurona artificial. Cada neurona posee entradas con pesos sinápticos específicos que determinan el valor a ser procesado en su interior mediante una función de activación que devuelve la salida de la neurona. La agrupación de neuronas conectadas entre sí determina una RNA. La misma se organiza en niveles denominados capas. Esta red junto con las interfaces de entrada y salida formarán el sistema global de aprendizaje.

Las RNA responden a un conjunto de estímulos de entradas (datos), recepcionados por la primera capa de neuronas, una regla de propagación de las señales hacia adelante y un valor final procesado por la última capa. Generalmente, en cuanto a su topología o estructura, las RNA distinguen tres tipos de capas de neuronas: de entrada, de salida y ocultas (que no tienen interacción con el exterior). Aprenden examinando los registros individuales, generando una predicción para cada registro y realizando ajustes a las ponderaciones cuando realizan una predicción, ya sea correcta o incorrecta. Este proceso se ejecuta reiteradas veces, mejorando sus predicciones hasta haber alcanzado uno o varios criterios de detenimiento.

C. Deep Reinforcement Learning

Al usar AR para resolver diversos problemas de toma de decisiones podemos recurrir a diferentes algoritmos, algunos de los más conocidos son *Q-Learning* o *Sarsa*. Pero en determinadas ocasiones, estamos en presencia de un entorno con muchos estados y acciones posibles que dificultan el proceso de aprendizaje y la ejecución de estos algoritmos por su carácter tabular.

DQN es el primer método de aprendizaje profundo desarrollado, propuesto por DeepMind en el año 2015 y que ha demostrado capacidades de convergencia en el aprendizaje [22]. El algoritmo combina AR con RNA; es decir, aproxima una función de valor de estado en el marco del *Q-Learning* con

ISBN: 978-987-88-2891-6

una red neuronal profunda. Esto permite utilizar el aprendizaje por refuerzo en casos más complejos y reales, donde el agente interactúa con un entorno que posee varios parámetros a tener en cuenta y la información es multidimensional, aprovechando las ventajas que proporcionan las RNA. En otras palabras, el algoritmo DQN permite generalizar el aprendizaje para trabajar con grandes espacios de estados; a la vez que permite trabajar con acciones discretas, lo que se ajusta directamente con el problema de manejo de materiales que más adelante detallaremos (Sec. III).

Por ejemplo, en el artículo original [22], el desarrollo se probó en diversos juegos Atari tomando varios fotogramas como vectores de estado de entrada, para cada acción como salida. Se demostró que una sola arquitectura puede aprender con éxito las políticas de control en diversos entornos con mínimo conocimiento previo, utilizando el mismo algoritmo e hiperparámetros en cada juego.

Uno de los elementos distintivos de este algoritmo es la utilización del mecanismo de *"Experience replay"*. Para cada iteración en el entrenamiento se almacena en memoria una muestra de la información obtenida durante la interacción del agente con el entorno. Esto permite que durante el aprendizaje se extraigan conjuntos de datos de la memoria al azar evitando problemas de correlación en la secuencia de datos, obteniendo así mayor estabilidad en el proceso. Por otra parte, el algoritmo utiliza dos RNA durante el entrenamiento y optimización. Por un lado, la Q-Network (*policy net*) para la estimación del valor de función actual, la que, a su vez, se optimiza teniendo en cuenta una red objetivo congelada (*target net*) la cual posee una actualización periódica con los últimos pesos obtenidos en cada paso. Esta última permite obtener los valores de función esperados y por lo tanto el cálculo del error (*loss*) respecto a la política. La actualización constante de los pesos de la red permite que el entrenamiento sea más estable al evitar oscilaciones a corto plazo.

Todas las características mencionadas permiten obtener un algoritmo de AR con mayor robustez, reduciendo la inestabilidad y divergencias que pudieran existir al utilizar una RNA para el procesamiento de datos y aprendizaje inteligente en entornos complejos.

III. Caso de Estudio

Como caso de estudio se toma una variante a la presentada en el reciente trabajo [23]. En el mismo, se plantea un problema de manejo de materiales, donde se busca minimizar las distancias recorridas, el uso de los medios de transporte y las demoras en las entregas que interrumpen el sistema productivo. Durante la resolución de un problema de este tipo se debe evitar la interrupción del flujo de trabajo asegurando la provisión en tiempo y forma de los insumos necesarios para las diferentes estaciones de trabajo, procurando lograr el normal funcionamiento del sistema de producción. A continuación se define los componentes del problema así como la conceptualización de cada uno de los elementos necesarios para plantear el problema de aprendizaje por refuerzos como un problema de decisión de Markov.

A. Definición del Problema

En el problema abordado, el sistema está compuesto por diez estaciones de trabajo ($S_i \ \forall i = 1, 2, ..., 10$) que consumen dos tipos de parte (P_1, P_2) cada una simultáneamente. En cada estación existen *buffers* de almacenamiento ($B_k \ \forall k = 1, 2, ..., 20$) para cada parte cuyas capacidades son iguales al doble de la cantidad de partes de cada tipo consumidas en una hora. Hay nueve vehículos de tres tipos diferentes ($V_{je} \ \forall j \in \{1, 2, 3\} \vee e \in \{1, 2, 3\}$) y se desarrollan distintos planes de manejo de materiales. Cada plan de manejo de materiales tiene asociada una ruta a seguir (W_c), un transporte específico (T_d), tipos y cantidades de partes a llevar a las estaciones. Los equipos de transporte no son todos iguales, es decir, cada tipo de vehículo V_{je} tiene su propia capacidad de carga y puede llevar cualquier tipo de parte.

B. Estado del Sistema

Para la representación del estado del sistema se consideran las magnitudes de tres de las variables, a saber: el tiempo de espera al cargar (T_d), el stock inicial (s_0) y la cantidad de viajes por vehículos (M), manteniéndose constantes los demás parámetros del sistema. De esta manera, el vector de estado del sistema queda definido como s_t = (T_d, s_0, M). Para la discretización del estado, el tiempo T_d se discretiza con un paso de 10 segundos en el intervalo [60, 300]; el stock s_0 puede tomar valores entre [10, 50] con un paso de 10 unidades; y la cantidad de viajes por vehículos debe encontrarse en el intervalo de enteros [1, 5]. El número final de estados o espacio de estados, por lo tanto, surge del producto cartesiano entre los subconjuntos de los posibles valores que toman estas tres variables comentadas, el cual es 625.

C. Acciones

Considerando que en el problema de decisión planteado el agente debe encontrar la combinación óptima de T_d, s_0 y M que minimice un resultado esperado; y sabiendo que el estado puede variar de uno de los 625 disponibles a cualquier otro sin inconvenientes, se definió que el espacio o vector de acciones sea coincidente con el de estados. Por lo tanto, se cuenta con 625 acciones posibles a elegir por el agente, permitiéndole seleccionar como próximo estado el mismo en que se encuentra u otro de los 624 restantes.

D. Función de Recompensa

En este problema, se busca optimizar el desempeño del sistema basándose en al grado de movimientos, uso de los *buffers* y tiempos de espera. Para poder evaluar esta mejora del plan, se evalúa los resultados de las decisiones escogidas por el agente mediante la siguiente función: $\mathscr{L}(u/s) = C_1 w_1 + C_2 w_2 + C_3 w_3$; siendo C_1 el porcentaje de ocupación de los transportes; C_2 la porción de tiempo en que los *buffers* se llenan completamente y C_3 el porcentaje de tiempo de espera total para consumir las partes. Los factores de peso w_1, w_2 y w_3 se fijaron en 0.3, 0.3 y 0.4, respectivamente; otorgando mayor importancia al retraso en la recepción de las partes, ya que esto generaría la interrupción del flujo de

ISBN: 978-987-88-2891-6

trabajo del sistema. Con base en lo anterior, se define la función de recompensa del algoritmo como $\mathscr{R} = 1/L(u/s)$. Es decir, la recompensa, la cual se pretende maximizar, aumenta al disminuir el resultado de la simulación, que desea minimizarse.

E. Deep Q-Learning Network

Como se planteó anteriormente, se aplica el algoritmo DQN al sistema, con el objetivo de alcanzar el mejor desempeño, basándose en la elección óptima del conjunto de variables de interés. La estructura del algoritmo se presenta a continuación.

Algoritmo DQN

1: Inicializar memoria replay
2: Inicializar función principal acción-valor Q con pesos random w
3: Inicializar función objetivo acción-valor $Q*$ con pesos $w*$=w
4: Cargar datos iniciales: estados S_t y acciones A_t
5: **for** episodio=0 **to** i **do**
6: Resetear estado inicial
7: **for** paso=0 **to** j **do**
8: Con probabilidad ϵ seleccionar una acción A_t random
9: Ejecutar acción y observar recompensa
10: Identificar próximo estado S_{t+1}
11: Guardar transición en memoria
12: Tomar una muestra random de la memoria
13: Calcular Q para S_t y S_{t+1}
14: Calcular Q esperada
15: Calcular la pérdida a partir de Q actual y la función Q esperada
16: **end for**
17: Actualizar $Q*$
18: Actualizar ϵ
19: **end for**

F. Implementación computacional

Para modelar el sistema y obtener un entorno para la ejecución del algoritmo, se decidió realizar el modelado computacional y simulación del piso de planta con el software *Tecnomatix Plant Simulation* (TPS) de Siemens. En la Fig. 1 se muestra el modelo de simulación del problema en cuestión.

Por medio de la librería pywin32 se conectó el simulador de eventos discretos usado a Python. Como ya se mencionó, la arquitectura de *Simulai* fue la que nos permitió el intercambio de información entre TPS y Python. La misma puede extenderse a *Robot Operating System* (ROS) para conectarse a algún dispositivo, como a otros simuladores de eventos discretos.

Cabe destacar, que la implementación del algoritmo DQN fue desarrollada en lenguaje Python. En dicho desarrollo, también se incluyó la librería Pytorch, la cual brinda un conjunto de herramientas optimizadas para el desarrollo de algoritmos de aprendizaje automático.

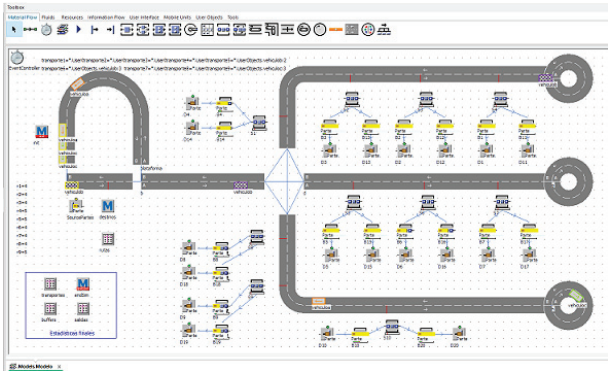

Fig. 1. Modelo de simulación en TPS.

TABLE I
PARÁMETROS PROBADOS EN EL ALGORITMO DQN.

Gamma	0.1	0.5	0.99	-
Learning Rate	0.0001	0.001	0.01	-
Batch Size	25	50	100	500
Target Update	50	100	500	1000

IV. RESULTADOS

El algoritmo DQN presentado, como todo algoritmo de aprendizaje por refuerzos, contiene una cantidad de hiperparámetros que deben ser definidos de antemano, tales como topología de las redes neuronales empleadas, tasa de aprendizaje, factor de descuento, entre otros.

En el algoritmo desarrollado, tanto la función política del agente como las funciones de valor se aproximan mediante RNAs, la cuales toman como entrada el vector de estado del sistema, mientras que la salida está definida con una función ReLu. A su vez, las mismas están definidas con dos capas ocultas, de 64 y 124 neuronas respectivamente, también empleando funciones de activación de tipo ReLU.

Una vez definidas las topologías de la redes se procedió a realizar diversas exploraciones, mediante prueba y error, haciendo uso de distintos parámetros de entrenamiento con el objetivo de identificar aquellos que otorgan un mejor desempeño del algoritmo. Para esto se llevaron a cabo 144 pruebas (Fig. 2), utilizando en cada una de ellas una combinación diferente para los valores de hiperparámetros dados en la Tabla I. Nótese que el hiperparámetro *gamma* define el nivel de aprendizaje del algoritmo en un horizonte de tiempo, el cual tiene en cuenta los elementos actuales ($\gamma = 0$) pero también los futuros ($\gamma = 1$) que podrían utilizarse para el entrenamiento. Por otro lado, el hiperparámetro de aprendizaje *learning rate*, permite definir la rapidez con que el código se adapta al problema en cuestión. En cada caso, se ejecutó el algoritmo durante 200 episodios de 100 pasos cada uno. Es importante tener en cuenta que el estado inicial en cada episodio es siempre aleatorio.

Para el entrenamiento de las redes neuronales utilizadas se usó el método iterativo de descenso de gradiente estocástico.

ISBN: 978-987-88-2891-6

Particularmente, se usó el algoritmo Adam [24], que permite la optimización mediante el cálculo del error entre los valores actuales del entrenamiento y una muestra de valores anteriores, modificando de esta forma los pesos del modelo mediante el algoritmo de retropropagación (*backpropagation*). La cantidad de pesos que se actualizan se determina por el valor del hiperparámetro en cuestión, siendo la mayoría de las veces determinante para el adecuado desempeño del algoritmo. A su vez, el tamaño del batch (*batch size*) y el nivel de actualización de la red objetivo (*target update*) definen parámetros propios del DQN como se definió anteriormente. El primero refiere a la cantidad de elementos que son recogidos de la memoria del algoritmo para lograr la *"Experience Replay"*. El otro indica cada cuantas iteraciones se actualiza la red objetivo.

Como se puede observar en la Fig. 2 las funciones de recompensa acumulada obtenidas para las distintas pruebas poseen una gran variación. Sin embargo, utilizando un alisado (*smoothing*) del 80% se observa una mayor densidad de líneas que reflejan un correcto aprendizaje del algoritmo (pendiente positiva), disminuyendo hacia desempeños totalmente ineficientes.

Esto demuestra cómo la selección de los hiperparámetros de entrenamiento puede condicionar el alcance de los objetivos cuando se utilizan estos métodos de aprendizaje.

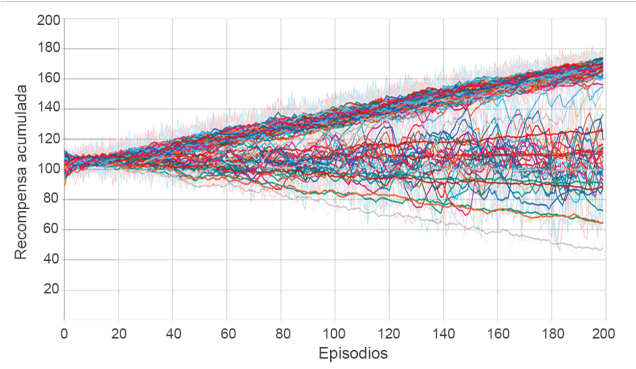

Fig. 2. Funciones de recompensa para la totalidad de pruebas realizadas.

A partir de esto, se seleccionó el conjunto de valores de parámetros que mejor desempeño logró, es decir, mayor recompensa obtuvo para el problema en estudio. En la Tabla II se muestran los parámetros seleccionados.

Por otra parte, es importante resaltar la utilización de una estrategia de exploración ϵ-*greedy*. La misma se basa en la utilización del parámetro épsilon (ϵ), el cual determina el nivel de exploración y explotación que desarrolla el algoritmo. Durante la fase de entrenamiento, el factor de exploración, ϵ, se disminuye linealmente, desde un valor inicial $\epsilon = 1$ hasta un valor final $\epsilon = 0.2$. De esta forma, inicialmente el algoritmo tiene un comportamiento puramente exploratorio y, a medida que transcurren los episodios de entrenamiento, la explotación comienza a prevalecer. El término explotación se refiere a la selección de la acción que presenta el mayor valor de función, es decir, se selecciona la acción con valor máximo para el

TABLE II
PARÁMETROS UTILIZADOS EN EL ALGORITMO DQN.

PARÁMETRO	VALOR	PARÁMETRO	VALOR
Epsilon Start	1	Batch Size	100
Epsilon End	0.2	Optimization	Adam
Gamma	0.1	Learning Rate	0.001
Target Update	50	Loss Function	MSELoss

episodio en que se encuentra, en lugar de elegir una al azar como se realiza durante la exploración.

Las pruebas con los parámetros seleccionados se realizaron para 200, 2000 y 5000 episodios. En las mismas se observó que el algoritmo converge a una solución y que el incremento en el número de épocas de entrenamiento no genera un resultado diferencial, más allá de los 300 episodios de entrenamiento. De esta manera, se procedió a entrenar el agente durante 2000 episodios con un decaimiento lineal del factor de exploración, tal que $\epsilon \in [1, 0.2]$, desde el inicio hasta los 200 episodios. Los resultados obtenidos se presentan en la Fig. 3.

En la Fig. 4 se presenta la evolución de la función de pérdida para los primeros 300 episodios de entrenamiento (por una cuestión de escala y mejor visualización solo se presenta este rango). Como puede apreciarse, el algoritmo logra un aprendizaje con una reducción exponencial de la pérdida, convergiendo rápidamente a una política óptima. Tal como se definió en la Tabla II, la función de pérdida utilizada es la MSELoss [25]. Ésta mide el error cuadrático medio entre un valor ingresado (valor de función del estado actual) y otro comparativo (valor de función esperada a partir de la red objetivo y al próximo estado).

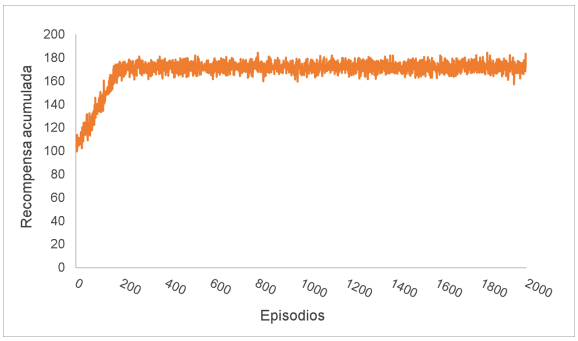

Fig. 3. Recompensa acumulada para 2000 epsiodios.

Finalmente, para comprobar y analizar el desempeño del algoritmo, se realizó el testing de los casos analizados. Para esto, se guardaron los modelos ya entrenados. Se probó el desempeño de la política obtenida para 30 estados iniciales elegidos aleatoriamente. Durante esta fase de prueba, se comprobó que el agente siempre elegía la acción que llevaba al sistema al mejor estado (óptimo), el cual es (T_d, s_0, M) = (300, 50, 1), con una recompensa $\mathscr{R} = 1,8926$. Es para destacar, que la política encontrada verifica su capacidad de

ISBN: 978-987-88-2891-6

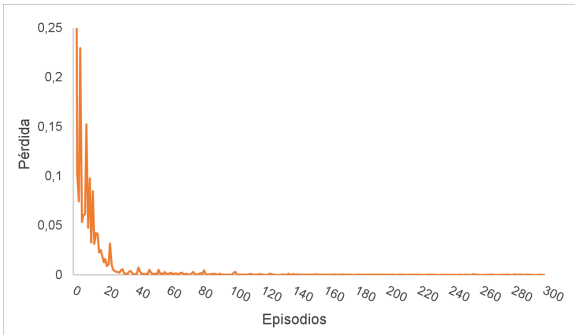

Fig. 4. Función de pérdida.

generalización dado que los estados iniciales aleatorios no fueron vistos durante la fase de entrenamiento.

V. Conclusión

La utilización de métodos de aprendizaje profundo utilizando RNA en combinación con entornos simulados resulta de gran ventaja para la optimización de procesos de manufactura dentro de los que hoy conocemos como Industria 4.0, donde la utilización e interpretación de los datos se vuelve primordial para la toma de decisiones segura en busca de alcanzar ciertos objetivos de los sistemas productivos.

En este trabajo abordamos un problema complejo de manejo de materiales y se utilizó el algoritmo de aprendizaje propuesto en combinación con un simulador de eventos discretos. El caso de estudio demuestra la funcionalidad del método, el cual permite minimizar los costos del manejo de materiales aprendiendo e identificando la combinación de variables que permiten hallar la mejor solución al problema planteado. Además, se evidencia cómo la selección de los parámetros de entrenamiento adecuados determina la eficiencia del algoritmo, influyendo directamente en la rapidez y convergencia del aprendizaje llevado a cabo por el agente en el entorno en que se desarrolla.

Finalmente, es para destacar la potencialidad de los desarrollos realizados, dado que permiten extender la metodología de toma de decisiones y procesamiento de información para plantas de manufactura que integren equipos de diferente naturaleza, ya sean máquinas herramientas, robots, elementos de transporte, personas, etc. Los desarrollos de este trabajo también contribuyen respecto a la integración de tecnologías para el desarrollo rápido y facil de lo que hoy se conoce como ”gemelos digitales”, dado que permite la interacción entre los equipos reales y sus homólogos simulados.

References

[1] Z. Shi, Y. Xie, W. Xue, Y. Chen, L. Fu, and X. Xu, “Smart factory in industry 4.0,” *Systems Research and Behavioral Science*, vol. 37, 06 2020.

[2] T. Zheng, M. Ardolino, A. Bacchetti, and M. Perona, “The applications of industry 4.0 technologies in manufacturing context: a systematic literature review,” *International Journal of Production Research*, vol. 59, 10 2020.

[3] E. Capawa Fotsoh, N. Mebarki, P. Castagna, and P. Berruet, *A Classification for Reconfigurable Manufacturing Systems*. Cham: Springer International Publishing, 2020, pp. 11–28. [Online]. Available: https://doi.org/10.1007/978-3-030-28782-5_2

[4] A. Kusiak, “Smart manufacturing,” *International Journal of Production Research*, vol. 56, no. 1-2, pp. 508–517, jan 2018. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/00207543.2017.1351644

[5] E. Goldratt, *What is this Thing Called Theory of Constraints and how Should it be Implemented?* North River Press, 1990. [Online]. Available: https://books.google.com.ar/books?id=FA8KAQAAMAAJ

[6] W. de Paula Ferreira, F. Armellini, and L. A. De Santa-Eulalia, “Simulation in industry 4.0: A state-of-the-art review,” *Computers and Industrial Engineering*, vol. 149, p. 106868, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835220305635

[7] S. Carta, A. Ferreira, A. S. Podda, D. R. Recupero, and A. Sanna, “Multi-dqn: An ensemble of deep q-learning agents for stock market forecasting,” *Expert Systems with Applications*, vol. 164, 2 2021.

[8] Y. Shi, W. Li, L. Zhu, K. Guo, and E. Cambria, “Stock trading rule discovery with double deep q-network,” *Applied Soft Computing*, vol. 107, 8 2021.

[9] T. Théate and D. Ernst, “An application of deep reinforcement learning to algorithmic trading,” *Expert Systems with Applications*, vol. 173, 7 2021.

[10] H. Park, M. K. Sim, and D. G. Choi, “An intelligent financial portfolio trading strategy using deep q-learning,” *Expert Systems with Applications*, vol. 158, 11 2020.

[11] C. Ma, J. Zhang, J. Liu, L. Ji, and F. Gao, “A parallel multi-module deep reinforcement learning algorithm for stock trading,” *Neurocomputing*, vol. 449, pp. 290–302, 8 2021.

[12] S. Guo, X. Zhang, Y. Du, Y. Zheng, and Z. Cao, “Path planning of coastal ships based on optimized dqn reward function,” *Journal of Marine Science and Engineering*, vol. 9, pp. 1–23, 2 2021.

[13] Y. Yang, L. Juntao, and P. Lingling, “Multi-robot path planning based on a deep reinforcement learning dqn algorithm,” *CAAI Transactions on Intelligence Technology*, vol. 5, pp. 177–183, 9 2020.

[14] J. Yun, Y. Goh, and J. M. Chung, “Dqn-based optimization framework for secure sharded blockchain systems,” *IEEE Internet of Things Journal*, vol. 8, pp. 708–722, 1 2021.

[15] S. Luo, L. Zhang, and Y. Fan, “Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning,” *Computers and Industrial Engineering*, vol. 159, p. 107489, 9 2021.

[16] M. Neves, M. Vieira, and P. Neto, “A study on a q-learning algorithm application to a manufacturing assembly problem,” *Journal of Manufacturing Systems*, vol. 59, pp. 426–440, 4 2021.

[17] J. Leng, C. Jin, A. Vogl, and H. Liu, “Deep reinforcement learning for a color-batching resequencing problem,” *Journal of Manufacturing Systems*, vol. 56, pp. 175–187, 7 2020.

[18] P. Colo, Pirozzo, and S. Sueldo, “Simulai,” 2020. [Online]. Available: https://simulai.readthedocs.io/en/latest/?badge=latest

[19] A. G. Sutton, R. S., & Barto, *Reinforcement learning: An introduction*. MIT Press, 1998.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016. [Online]. Available: https://books.google.co.in/books?id=Np9SDQAAQBAJ

[21] B. M. del Brío and A. S. Molina, *Redes neuronales y sistemas borrosos : introducción teórica y práctica*, 3rd ed. Alfaomega Grupo Editor S.A., 2006.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2 2015.

[23] S. Govindaiah and M. D. Pey, “Applying reinforcement learning to plan manufacturing material handling part 1: Background and formal problem specification,” *ACMSE 2019 - Proceedings of the 2019 ACM Southeast Conference*, pp. 168–171, 4 2019.

[24] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[25] Pytorch, “Torch Contributors,” 2019. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html