ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO Y SIMULACIÓN DE UN ALGORITMO DE DECISIÓN DE CAMINO PARA UN ROBOT DE LABERINTO UTILIZANDO REDES NEURONALES

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y AUTOMATIZACIÓN

JESSICA PAMELA BALSECA BALSECA

jessica.balseca@epn.edu.ec

DIRECTOR: DR. GEOVANNY DANILO CHÁVEZ GARCÍA

danilo.chavez@epn.edu.ec

DMQ, enero 2022

CERTIFICACIONES

Yo, Jessica Pamela Balseca Balseca declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

JESSICA PAMELA BALSECA BALSECA

Certifico que el presente trabajo de integración curricular fue desarrollado por Jessica Pamela Balseca Balseca, bajo mi supervisión.

DR. GEOVANNY DANILO CHÁVEZ GARCÍA
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el producto resultante del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

JESSICA BALSECA

DR. DANILO CHÁVEZ

NOMBRE_COLABORADOR(ES)

DEDICATORIA

A Dios por haberme dado el regalo más grande de tener a mi hermosa familia y a través de sus bondades ha derramado salud, sabiduría, y valentía sobre la misma.

Con todo mi corazón este sueño que hoy se cumple se lo dedico principalmente a mis padres Alonso y Elva por su esfuerzo incansable todo su amor y su apoyo han forjado en mí la persona que soy ahora.

A mis hermanitas pequeñas Camila y Erika por hacer de mi vida más plena y dichosa, por acompañarme con todo su apoyo y cariño en esta etapa. Espero ser un ejemplo para ustedes y que encuentren en mí una gran amiga.

A mis mascotas por ser la fuerza de cada mañana y enseñarme el significado del amor puro Lucas, Pepe, Pocho y Ponky, gracias por ser la compañía de cada madrugada.

A la memoria de una mujer ejemplar que en su corta vida ha dejado un legado muy grande y ha sido fuente de inspiración para muchas a no dejar de soñar. Con amor, para Selena Quintanilla Pérez.

AGRADECIMIENTO

A Dios que de principio a fin me previó la sabiduría y el valor necesario para terminar con esta etapa y hacer de un sueño una realidad.

A mi padre por ser gran parte de mi alma, por tu paciencia, por tu amor y tu apoyo incondicional, porque a través de las palabras correctas me diste fuerza para no rendirme en varios momentos difíciles y a pesar de tu ocupación siempre tuviste un espacio para ocuparte de mí. TE AMO PAPÁ, eres el mejor.

A mi madre Elva, mis hermanas Erika y Camila porque siempre fueron un consuelo y una fuerza inquebrantable para seguir con este sueño. Gracias por llevarme en sus oraciones y en sus corazones en todo momento.

A mis padrinos que han sido parte de mi familia y me educaron desde pequeña, gracias por el amor y apoyo como segundos padres, para Mercedes, Neil, Jorge y Margia.

A la Escuela Politécnica Nacional por recibirme en sus aulas y darme el orgullo de pertenecer a esta prestigiosa institución para proporcionarme conocimientos valiosos y sobre todo por hacer de mí una mujer con un carácter forjado y de voluntad.

Al PhD. Danilo Chávez por su paciencia y darme la oportunidad de llevar a cabo este trabajo de titulación, tanto como persona y como maestro es un ejemplo a seguir.

A la Unidad de Mantenimiento Electrónico por permitirme ser parte de esta hermandad, su amistad y apoyo han hecho de la Universidad un mejor lugar para compartir y crecer como personas tanto personal como profesionalmente.

A mis amigos por creer en mí, por recordarme constantemente de mis capacidades para lograr mis objetivos y sobre todo el amor que me han demostrado en este tiempo, especialmente a: María José, David, Gissela, Jorge Andrés, Pamela, Mónica, Anabel, Galo, Paulina, Paola y Aracely.

Finalmente, a quienes me apoyaron con sus conocimientos y me recibieron de la mejor manera en el ámbito profesional para culminar con esta etapa al Ing. Santiago Muñoz y al equipo de mantenimiento Pronaca Puembo.

ÍNDICE DE CONTENIDO

RESUMEN

El presente trabajo consiste en el diseño y simulación de un algoritmo de decisión capaz de resolver laberintos empleando redes neuronales bajo las arquitecturas secuencial y recursiva LSTM. Previo al diseño, se definen algunos conceptos generales e importantes sobre el aprendizaje profundo por refuerzo junto con los componentes y el algoritmo de aprendizaje Q-learning y algoritmo de optimización Adam con la función de pérdida MSE. Se considera un enfoque mixto de tipo cualitativo ya que se necesita profundizar los conocimientos de algunas librerías compatibles con Python como NumPy encargada de visualizar datos mediante arrays, PyTorch enfocado al uso de redes neuronales, Matlplotlib que permite crear visualizaciones de los laberintos, Jupiter que ayuda a configurar modelos de las redes y el paquete Tkinter con el cual se realiza la interfaz entre usuario y máquina y Jupiter. En segunda instancia, se utiliza un enfoque cuantitativo ya que se construye el algoritmo Q-learning basado en los estados y acciones que realiza el agente para definir la recompensa en función de los objetivos a corto y largo plazo. Además, se genera laberintos aleatorios con el propósito de que el agente no sea capaz de memorizar el camino directo hacia la meta, por tanto, se desarrolla la repetición de la experiencia para que sea capaz de entrenarse ante esta diversidad. Por consiguiente, se realizaron varias pruebas que reflejan el entrenamiento, función de pérdida y comparación de arquitecturas, siendo la secuencial más efectiva proporcionando un valor superior al recursivo del 71% para 100 laberintos generados.

PALABRAS CLAVE: aprendizaje por refuerzo profundo, aprendizaje Q, laberintos, red neuronal secuencial, red neuronal recursiva.

ABSTRACT

The present job consists in the design and simulation of a decision algorithm capable of solving labyrinths using neural networks under the sequential and recursive architectures LSTM. Prior to design, some general and important concepts on deep reinforcement learning are defined along with the components and learning algorithm Q-learning and Adam optimization algorithm with the loss function MSE. It is considered a mixed approach of qualitative type since it is necessary to deepen the knowledge of some libraries compatible with Python as NumPy in charge of visualizing data through arrays, PyTorch focused on the use of neural networks, Matlplotlib that allows you to create visualizations of the mazes, Jupiter that helps to configure models of the networks and the Tkinter package with which the interface between user and machine is made and Jupiter. In the second instance, a quantitative approach is used since the Q-learning algorithm is constructed based on the states and actions performed by the agent to define the reward based on short- and long-term objectives. In addition, random labyrinths are generated with the purpose that the agent is not able to memorize the direct path to the goal, therefore, the repetition of the experience is developed to be able to train before this diversity. Therefore, several tests were performed that reflect the training, loss function and comparison of architectures, being the most effective sequential providing a value higher than the recursive 71% for 100 generated labyrinths.

KEYWORDS: deep reinforcement learning, Q learning, labyrinths, sequential neural network, recursive neural network.

1 INTRODUCCIÓN

En los últimos años se ha visto a los robots móviles cada vez más arraigados en la sociedad moderna ocupando cada vez una mayor cantidad de aplicaciones recreacionales como el recientemente lanzado Xiaomi Cyberdog, el cual es un robot en forma de perro que no cumple otra función que alegrarnos el día ante el llamado del ser humano [1]. Existen otro tipo de robots denominados Roomba que están orientados a la limpieza domestica [2] o incluso cumpliendo tareas de empaque en las bodegas de Amazon, donde han conseguido realizarse en menos de una hora; lo que años atrás se tomaba más de un día completo [3]. Para poder realizar las actividades de los robots mencionados se utiliza la Inteligencia Artificial IA, esta es una rama de las ciencias de la computación involucrada en la creación de programas informáticos capaces de demostrar inteligencia [4].

Tradicionalmente cualquier software que muestra habilidades cognitivas como la percepción, la búsqueda, la planificación y el aprendizaje se considera parte de la IA. El aprendizaje por refuerzo consiste en aprender o ver la manera de hacer un mapeo de situaciones con el objetivo de maximizar una señal de recompensa numérica. Cuando se aplica el aprendizaje por refuerzo profundo se tiene una aproximación a la inteligencia artificial, la cual contiene una propiedad de aprendizaje basado en la prueba y error, gracias al uso de la retroalimentación. En un estado particular, la función de recompensa por una acción correcta puede no ser reconocida de forma instantánea [5]. Por ejemplo, un estudiante (agente) tiene la opción entre estudiar para aprobar unos exámenes o ir a jugar, esta última podría dar una recompensa instantánea en la forma de un pico de adrenalina pero que posiblemente en el futuro no sea una recompensa muy buena. Por lo que es importante definir una función de recompensa que haga un balance entre objetivos de corto y largo plazo.

En el presente trabajo se plantea aplicar inteligencia artificial para resolver laberintos, similares a los que se establecen en las competencias del Concurso Ecuatoriano de Robótica CER, es decir, los robots deben ser capaces de recorrer y salir del laberinto en el menor tiempo posible y de forma idónea [6]. Además, se utilizará métodos de aprendizaje de máquina a través de redes neuronales, el cual busca representaciones de los datos de entrada en un espacio de posibilidades predefinidos mediante una guía de señal de alimentación [7]. El método propuesto no es un algoritmo programado para seguir reglas preestablecidas, sino que aprende a través de refuerzos cuyo propósito es resolver el problema de toma de decisiones de camino en el laberinto.

1.1 OBJETIVO GENERAL

El objetivo general de este Proyecto Técnico es: diseñar y simular un algoritmo de decisión de camino para un robot de laberinto utilizando redes neuronales.

1.2 OBJETIVOS ESPECÍFICOS

Los objetivos específicos del Proyecto Técnico son:

1. Realizar una recopilación de información bibliográfica analizando las aplicaciones de las redes neuronales en robots móviles.
2. Diseñar un algoritmo de Deep Reinforcement Learning para toma de decisiones utilizando redes neuronales.
3. Realizar una simulación con laberintos generados aleatoriamente donde se podrá aplicar los algoritmos de resolución de laberinto.
4. Implementar una interfaz gráfica para presentar la resolución de un laberinto por los diferentes algoritmos.

1.3 ALCANCE

- Se hará una recopilación de información acerca de las redes neuronales aplicadas en robots móviles, además se realizará un análisis de arquitecturas de red neuronal secuencial y recursiva, y la forma en que se podrían aplicar en la toma de decisiones del robot de laberinto.
- Se diseñará un algoritmo de Deep Reinforcement Learning para toma de decisiones en el laberinto, se implementará una red neuronal profunda con arquitectura secuencial y otra con arquitectura recursiva, se detallará cuáles son sus entradas, sus salidas, el número de capas ocultas y el número de neuronas en cada una de estas. Estas redes deberán mapear la posible recompensa de escoger las opciones de camino disponibles en cada intersección del laberinto.
- Se definirá una función de recompensa que mida el avance de la resolución de un laberinto para realizar el entrenamiento de la red neuronal, haciendo balance entre objetivos de corto y largo plazo.
- Se desarrollará un algoritmo para generar laberintos de forma aleatoria en los que se pueda probar los algoritmos de resolución de laberinto.
- Se entrenará a las redes neuronales en los laberintos generados aleatoriamente utilizando la función de recompensa definida previamente.

- Se implementará una interfaz en Python donde se podrá visualizar los resultados obtenidos por los diferentes algoritmos para compararlos.

1.4 MARCO TEÓRICO

Esta sección engloba una revisión teórica sobre el aprendizaje profundo por refuerzo DRL con sus componentes, redes neuronales y arquitecturas de tipo secuencial y recursiva.

1.4.1 APRENDIZAJE PROFUNDO DL

El aprendizaje profundo “Deep Learning” es un subcampo del Machine Learning que a su vez pertenece al ámbito de la Inteligencia Artificial IA, tal como se muestra en la Figura 1.1. Cualquier tipo de software que contenga destrezas cognitivas como la percepción, planificación, búsqueda y aprendizaje se considera una IA. Sin embargo, no todos los programas pueden aprender, por lo que se incluyen algoritmos informáticos capaces de resolver problemas aprendiendo de los datos ya sea de forma supervisada, no supervisada y por refuerzo [4].

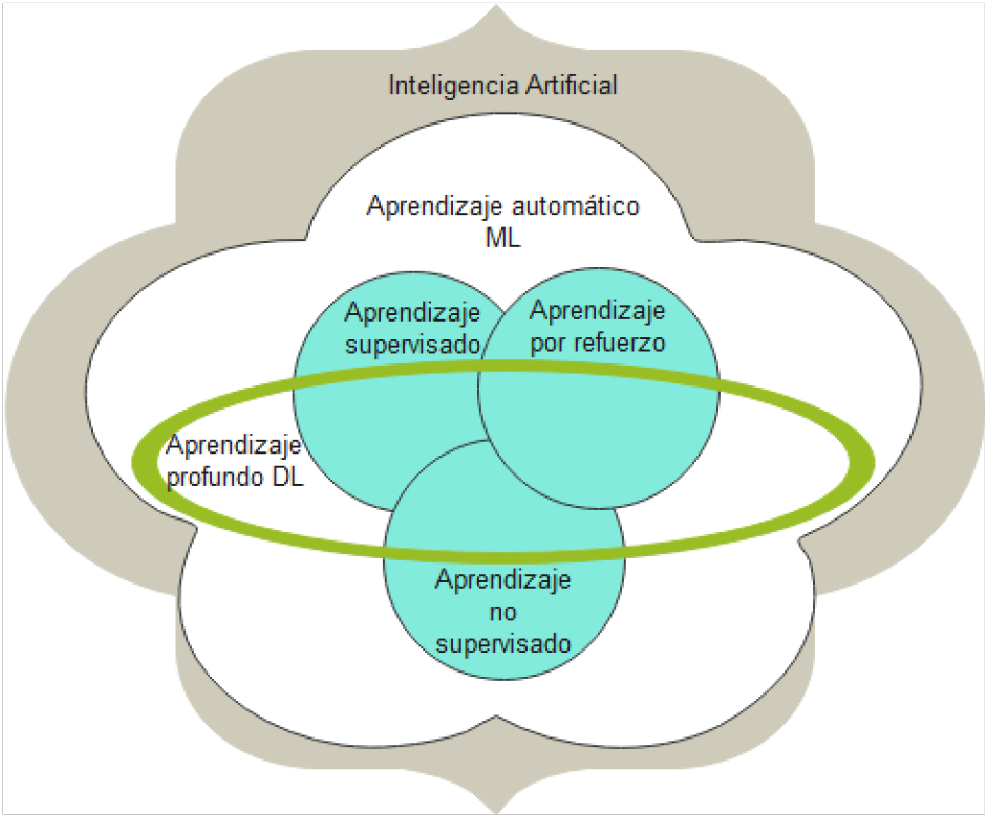


Figura 1.1. Ramas del aprendizaje automático

El aprendizaje profundo utiliza redes neuronales, el cual está compuesto desde decenas hasta centenas de capas ocultas y en forma sucesiva o secuencial. Además, el propósito de este aprendizaje consiste en brindar una información depurada de los datos, tal como

sucede en el ejemplo de la Figura 1.2, donde se realiza un reconocimiento de las imágenes para determinar el personaje de una base de datos [7].

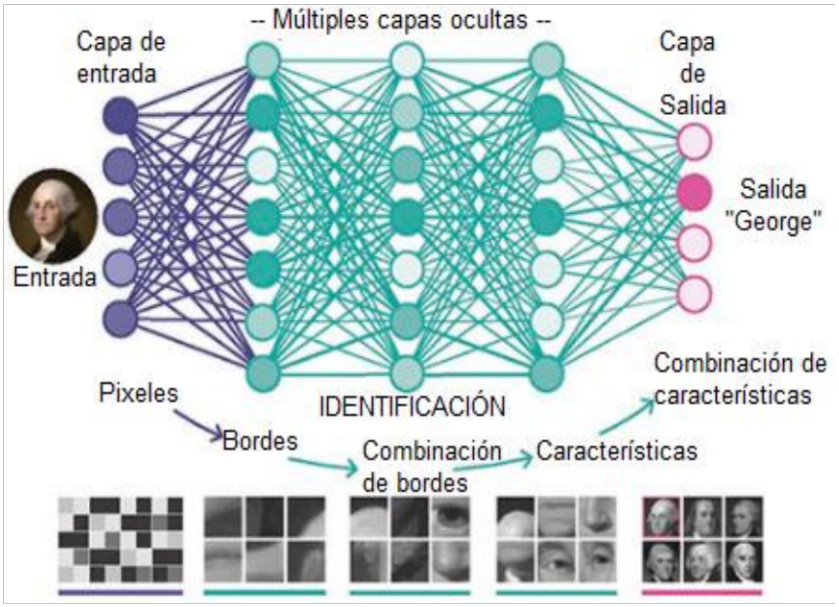


Figura 1.2. Redes neuronales de aprendizaje profundo [8]

1.4.2 APRENDIZAJE POR REFUERZO PROFUNDO DRL

Los programas de aprendizaje por refuerzo profundo aplican casos de prueba y error mediante una aproximación de funciones no lineales para resolver problemas en condiciones de incertidumbre. Además, utiliza programas informáticos o también denominados agentes, el cual tiene como función única tomar decisiones. Por ejemplo, cuando un robot recoge objetos, el algoritmo que toma la decisión de coger dicho objeto es el agente; mientras que el brazo del robot, el viento, entre otros; se denomina medio ambiente, tal como se muestra en la Figura 1.3 [4].

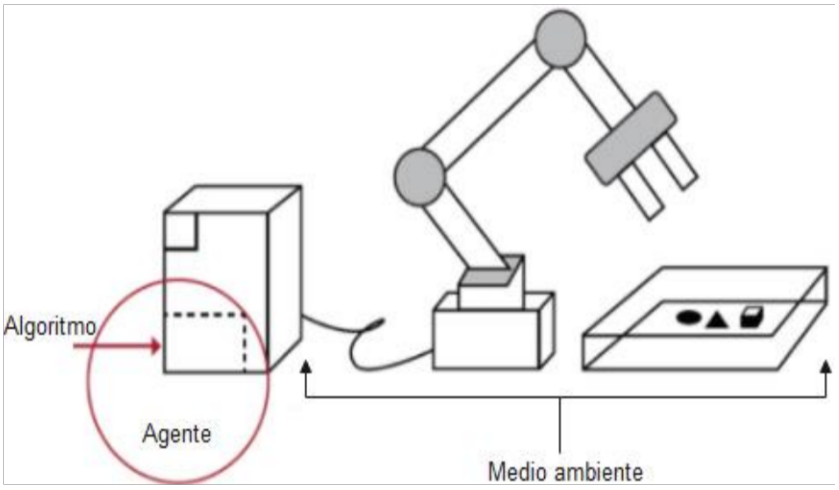


Figura 1.3. Aprendizaje por refuerzo profundo [4]

Por último, el DRL tiene como fortaleza aprender habilidades sencillas mediante técnicas de generalización. El desempeño de estas técnicas, trucos y nuevos consejos ayudan al entrenamiento profundo de la red. Sin embargo, no presenta una alta eficiencia ya que necesita millones de muestras para ofrecer un buen rendimiento [4].

Existe un método capaz de aprender valores de acción óptimos denominados Q-learning que tiene como propósito utilizar un algoritmo capaz de predecir el valor de un par de datos estado S_t y acción a_t para luego, compararlas con las recompensas observadas acumuladas R_{t+1} en un momento posterior a fin de actualizar los parámetros del algoritmo $Q(S_t, a_t)$. Todo este proceso se lleva a cabo para realizar mejores predicciones la siguiente vez; por tanto, el valor Q se actualiza según el valor actual más la cantidad del valor que se espera a futuro Q_F, visto en la Ecuación 1.1 [9].

$$\begin{cases} Q_F = \propto \left[R_{t+1} + \gamma * max\big(Q(S_{t+1}, a)\big) - Q(S_t, a_t)\right] \\ Q(S_t, A_t) += Q_F \end{cases} \quad (1.1)$$

Donde [9]:

- γ y $\propto$ son hiperparámetros que influyen en el modelo de aprendizaje, estas son variables que no dependen del aprendizaje real.
- $\propto$ representa la tasa de aprendizaje que se usa para el entrenamiento de la red, eso quiere decir ante valores pequeños realiza bajas actualizaciones en cada paso.
- El factor de descuento γ es un parámetro variable entre 0 y 1 que controla la disminución ante futuras recompensas en base a las decisiones tomadas por el agente.
- El agente elige una acción a al toma la decisión, primero un valor +1 y luego, 0 recompensas.

1.4.3 COMPONENTES DRL

Existen dos elementos principales del aprendizaje DRL, el agente que toma las decisiones basados en tres pasos indicados en la Figura 1.4 como evaluar el comportamiento, recopilar datos para el aprendizaje e interacción con el problema a fin de obtener un mejor desempeño. El segundo elemento lo representa el medio ambiente que reacciona ante las acciones del agente. Por tanto, el aprendizaje DRL busca la forma de asignar situaciones a acciones a fin de maximizar una señal de recompensa numérica [4].

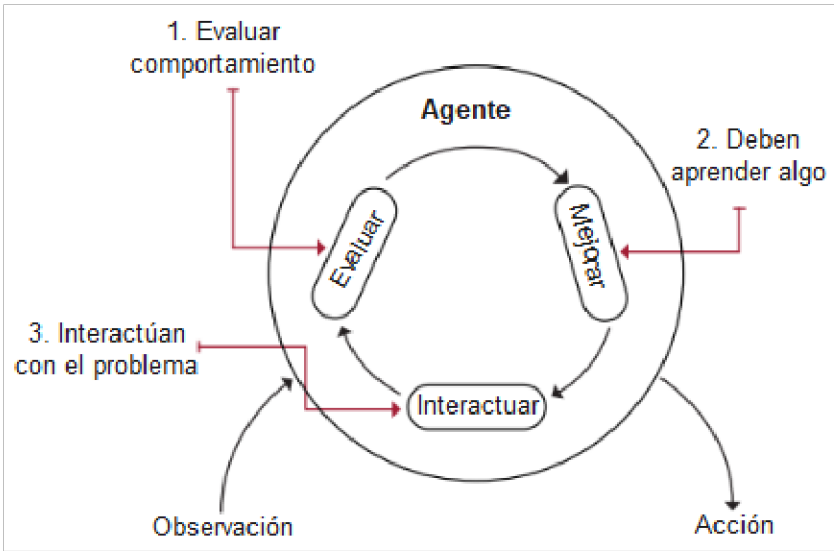


Figura 1.4. Pasos internos del agente [4]

Adicional a los dos componentes mencionados, existen 4 subelementos de un sistema DRL, tales como [5]:

- Una política que define el comportamiento del agente en un momento dado. Por tanto, puede abarcar funciones simples o complejas y tareas o procesos de búsqueda. En general, las políticas involucran ser estocásticas, brindando probabilidades para cada acción.

- La señal de recompensa establece el objetivo del problema DRL. En cada instante de tiempo, se envía un único número desde el entorno hacia el agente, donde maximiza la recompensa total. Por tanto, la señal de recompensa precisa todos los eventos malos y buenos para el agente.

- Luego, se utiliza una función de valor para determinar a largo plazo un evento bueno de la señal de recompensa, denominado valor de estado. En otras palabras, los valores son estimaciones de las recompensas. Sin embargo, son de gran preocupación al momento de toma una decisión, por lo que se busca acciones de mayor valor para obtener una mejor recompensa a largo plazo.

Por último, se tiene el modelo del entorno, el cual realiza inferencias del comportamiento del medio ambiente. El modelo tiene la capacidad de predecir el siguiente estado resultante y próxima recompensa cuando tiene como datos iniciales, el estado y la acción.

1.4.4 REDES NEURONALES

Las redes neuronales artificiales RNA representan un conjunto de algoritmos capaces de obtener un valor nuevo a la salida de la neurona, mediante la interconexión de las entradas y una combinación $f(x)$. A través de la Figura 1.5 se observa como una neurona posee un

conjunto de entradas denotadas como $X = \{x_1,\ x_2, \ldots, x_n\}$, estas ponderadas por el conjunto de pesos $W^i = \{W^i{}_1,\ W^i{}_2, \ldots, W^i{}_n\}$. El valor resultante se procesa bajo una función de activación, la cual modula los valores de entrada para generar una salida y_i [10].

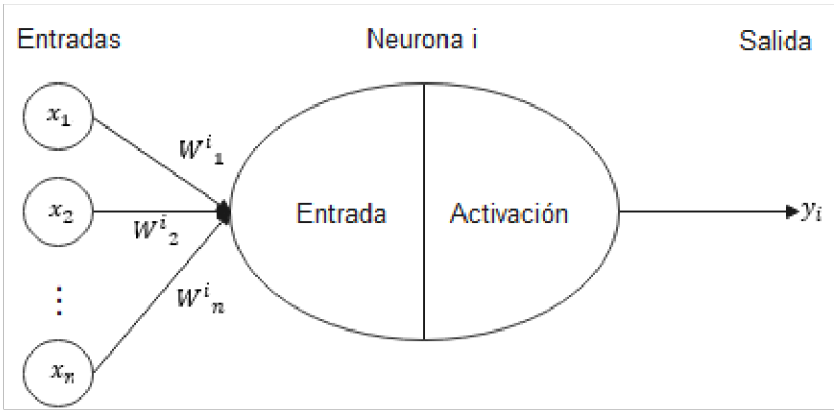


Figura 1.5. Redes neuronales

La función de entrada $z(x)$ se calcula en base a la combinación de la entrada x_j y el peso w_j^i ya sea determinando máximos, mínimos, sumatorios o funciones lógicas matemáticas, indicados en las Ecuaciones 1.2 a la 1.6 [10].

- Función suma ponderada.

$$z(x) = \sum_{j=1}^{n} x_j w_j^i \tag{1.2}$$

- Funciones máxima o mínima.

$$z(x) = max\left(x_1 w_1^i, x_2 w_2^i, .., x_n w_n^i\right) \tag{1.3}$$

$$z(x) = min\left(x_1 w_1^i, x_2 w_2^i, .., x_n w_n^i\right) \tag{1.4}$$

- Función lógica AND (∧) u OR (∨) cuando se trate de entradas binarias.

$$z(x) = \left\{x_1 w_1^i \ \wedge\ x_2 w_2^i \ \wedge\ x_n w_n^i\right\} \tag{1.5}$$

$$z(x) = \left\{x_1 w_1^i \ \vee\ x_2 w_2^i \ \vee\ x_n w_n^i\right\} \tag{1.6}$$

Por consiguiente, la función de entrada pasa por un proceso de activación o transferencia, modificando su valor antes de pasarlo a la salida. Existen diversas funciones de activación indicadas en las Ecuaciones 1.7 a la 1.11 [10].

- Función escalón, donde $\propto$ es el valor límite de activación.

$$y(x) = \begin{cases} 1 & si\ x\ \geq \propto \\ -1\ o\ 0 & si\ x < \propto \end{cases} \tag{1.7}$$

- Función lineal, donde β representa la pendiente de la transferencia.

$$y(x) = \beta x \tag{1.8}$$

- Función sigmoide, donde γ es un parámetro que suaviza el dato de salida.

$$y(x) = \frac{1}{1 + e^{-x/\gamma}} \tag{1.9}$$

- Función tangente hiperbólica.

$$y(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{1.10}$$

- Función rectificadora ReLU, siendo esta la más utilizada ya que muestra valores positivos [10].

$$y(x) = max(0, x) \tag{1.11}$$

1.4.5 ARQUITECTURAS DE REDES NEURONALES

La topología de una red neuronal consiste en tres elementos básicos en la configuración, tales como las capas de entrada, neurona o capas ocultas y la salida. Existen diversas arquitecturas de la red, siendo la más básica observada en la Figura 1.5 y otras de tipo secuencial, recursiva, etc.

1.4.5.1 Red neuronal secuencial

Las redes neuronales de tipo secuencial se considera como a una extensión de la arquitectura clásica ya que se basan en el siguiente proceso de inferencia [11]:

- El modelo selecciona diferentes asignaciones en una entrada, por lo que se asigna un espacio de representación.

- Luego, a la salida se escoge otro mapeo similar al primer proceso de la red neuronal $(n_1,\ n_2,\ n_3,\ n_4)$ y así sucesivamente hasta completar la predicción. Por tanto, en la Figura 1.6 se muestra una arquitectura secuencial base compuesto de 8 nodos, donde la predicción final se calcula con base a la siguiente función: $f_{n6,n8}\left(f_{n2,n6}\left(f_{n1,n2}(x)\right)\right)$.

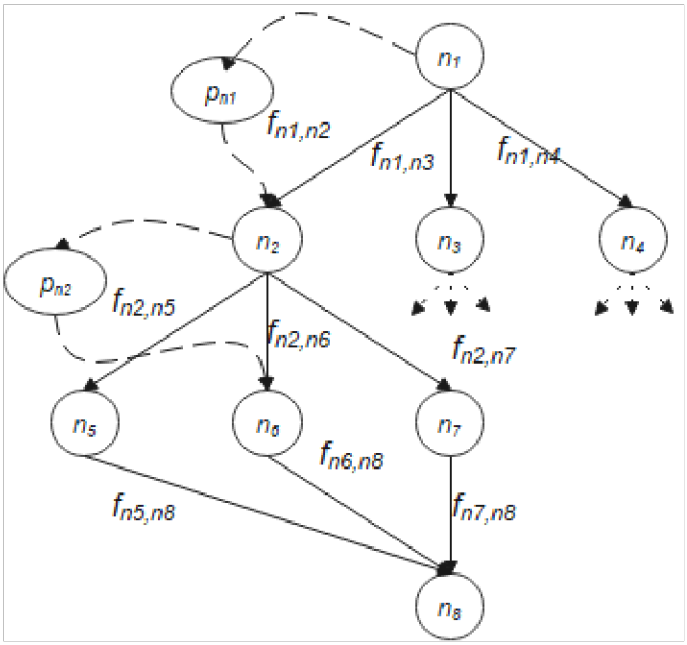


Figura 1.6. Red neuronal secuencial

1.4.5.2 Red neuronal recursiva

La red neuronal recursiva tiene como propósito compartir pesos en cada nodo por lo que se considera una red generalizada de tipo recurrente [12]. En general, la salida de la capa posterior puede ser conectada hacia las capas anteriores, tal como se presenta en la Figura 1.7. De esta manera, el proceso de una red toma como información su propio registro para generar un nuevo resultado [10].

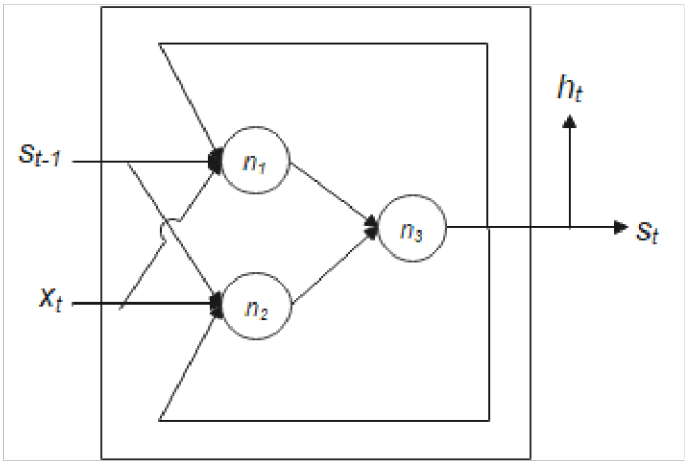


Figura 1.7. Red neuronal recursiva

En las redes recurrentes se menciona las celdas, cuya definición nace de la composición mutua de entradas y salidas a diferentes valores de la secuencia. En el ejemplo de la Figura 1.7 se denota que el vector de entrada x_t pertenecen a una dimensión $\mathbb{R}^m$ y la salida en cada paso de la recurrencia es de dimensión l por lo que se tiene un valor $h_t \in \mathbb{R}^l$. Además,

se establece un valor n que representa el estado de la red, entonces $s_t \in \mathbb{R}^n$, por consiguiente, la celda se expresa bajo la Ecuación 1.11 [10].

$$\begin{array}{rccc} f: & \mathbb{R}^{m+n} & \rightarrow & \mathbb{R}^{n+l} \\ & (x_t, s_{t-1}) & \rightarrow & (x_t, s_t) \end{array} \tag{1.12}$$

La red neuronal recurrente depende del número de entradas y salidas que sean colocadas, ya sea para obtener una única salida tal como se presenta en la Figura 1.8.a) o una única entrada con salida de varios pasos visto en la Figura 1.8.b). Por último, se observan en las Figuras 1.8.c) y d) como la red contiene varias entradas y salidas. Cabe mencionar que existen dos entradas y dos salidas por cada celda, no obstante, cuando no se refleja alguna entrada y/o salida es porque su valor es nulo [10].

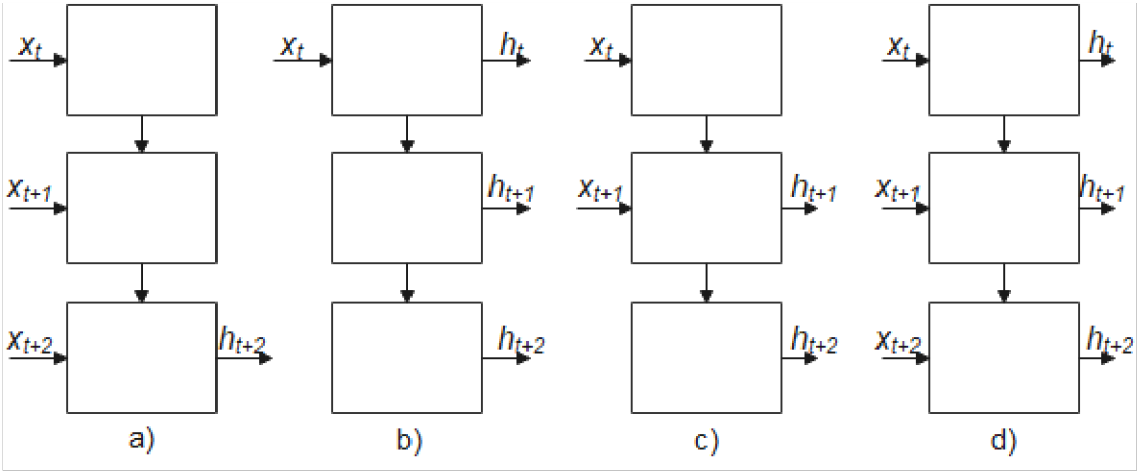


Figura 1.8. Tipos de entradas y salidas de las redes neuronales recurrentes

Además, existen numerosas arquitecturas de celdas, siendo las más utilizadas una memoria a corto plazo LSTM y la unidad recurrente cerrada GRU. La memoria LSTM tiene la capacidad de aprender dependencias desde puntos cercanos y alejados de una secuencia, gracias al canal de memoria que es una línea horizontal directa (parte superior de la celda), visto en la Figura 1.9 y al control del flujo de información del estado [10].

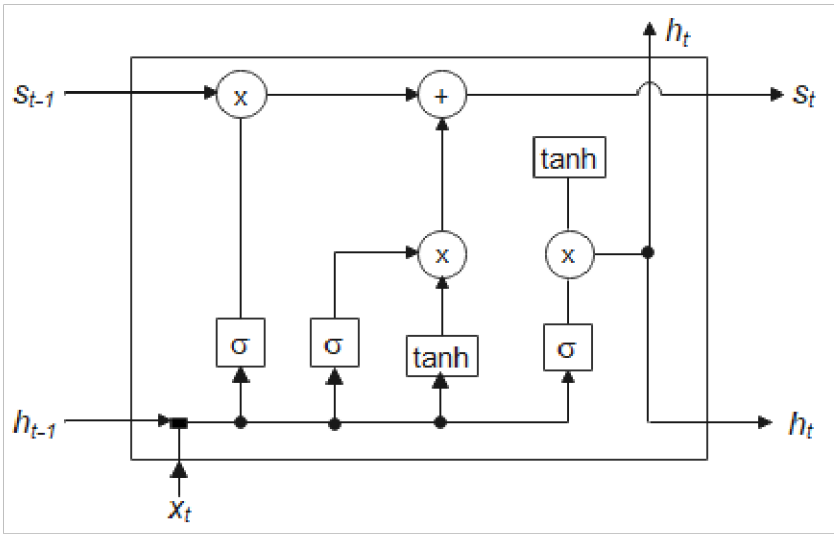


Figura 1.9. Celda LSTM

La red LSTM emplea una puerta del olvido f_t que consiste en realizar una observación sobre los valores de entrada x_t y salida del paso anterior h_{t-1} para decir cual parte de la memoria se desea conservar. El cálculo de f_t se desarrolla por medio de la Ecuación 1.12, el cual posee una misma longitud de los vectores de estado, b_f como vector de sesgos y una matriz de parámetros W_f producto de la concatenación entre los valores h_{t-1} y x_t [10].

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big) \tag{1.13}$$

Existe una puerta de entrada análoga i_t que controla la información que se añade a la memoria de la red, cuyo cálculo se determina a través de la Ecuación 1.13; donde W_i y b_i es la matriz de parámetros y vector de sesgos; respectivamente [10].

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{1.14}$$

Una vez que se determina el nuevo estado se obtiene la respuesta de salida de la red h_t, observando el valor de entrada y la salida anterior tal como se indica en la Ecuación 1.14 [10].

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t \odot tanh(s_t) \end{aligned} \tag{1.15}$$

Por simple inspección, se observa que las Ecuaciones anteriores utilizan funciones de activación ya sea con la tangente hiperbólica $tanh$, la función sigmoide σ y el producto Hadamard $\odot$. Esto implica que las puertas pueden contener valores difusos en el rango $[0,\ 1]$ para enviar cierto tipo de información [10].

En cambio, la arquitectura GRU (ver Figura 1.10) es una simplificación de la red LSTM que contiene dos salidas iguales debido a la fusión entre la respuesta y el estado, este último es una media ponderada del estado anterior y la información nueva. Este tipo de arquitectura se emplea cuando no se requiere una gran cantidad de datos para el entrenamiento de la red. De tal manera, que solo necesita de dos puertas para el flujo de información, una de reset encargada de seleccionar en concreto la información, por lo que se obtiene un vector r_t, visto en la Ecuación 1.15 [10].

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{1.16}$$

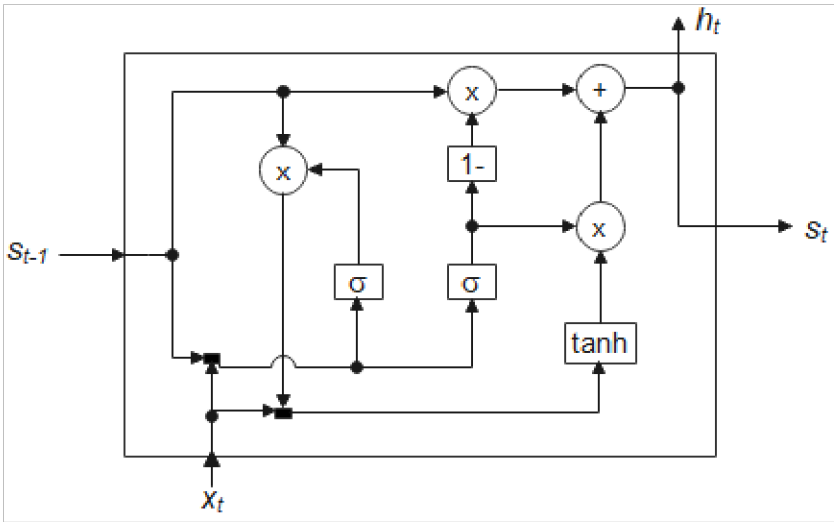


Figura 1.10. Celda GRU

Luego, posee una puerta análogo de control para mantener cierta información en la memoria, caso contrario borrarla. Por tanto, la salida se representa bajo la expresión de la Ecuación 1.16; donde W_z y W_s son matrices de parámetros, b_z y b_s son vectores de sesgo [10].

$$\begin{gathered} h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{S}_t \\ z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \\ \hat{S}_t = tanh(W_s \cdot [r_t \odot h_{t-1}, x_t] + b_s) \end{gathered} \tag{1.17}$$

En conclusión, las redes neuronales recurrentes son de bajo costo y tienen un buen funcionamiento cuando se trabaja en tareas complejas con un alto volumen de datos. Sin embargo, al aumentar el volumen de los datos se hace más compleja la red y se demora más en el tiempo de aprendizaje, por lo que se requiere una alta capacidad computacional para el preprocesamiento de los datos [13].

1.4.6 ALGORTIMOS DE OPTIMIZACIÓN

Durante el proceso de aprendizaje se calculan los pesos mediante cualquier algoritmo de optimización cuyo propósito es disminuir el error cometido en la predicción del modelo de la red. A continuación, se describen algunos algoritmos de optimización:

- El descenso de gradiente estocástico SGD es un método que actualiza los parámetros para reducir el error de predicción de la red. Para ello, realizar una propagación hacia adelante y atrás de cada registro para converger por lo que los mínimos de errores globales se vuelven muy ruidosos [14].
- La tasa de aprendizaje adaptativo Adadelta es un método dinámico que no necesita de un ajuste manual de la tasa de aprendizaje. Sin embargo, posee una sobrecarga computacional mínima superior al que posee el descenso de gradiente estocástico SGD y contiene una información de gradiente robusta [15].
- El algoritmo de gradiente adaptativo Adagrad emplea diferentes tasas de aprendizaje por cada parámetro que se actualiza durante el entrenamiento [14].
- El algoritmo de Adam es un método que reemplaza al SGD ya que actualiza los pesos en función de los datos de entrenamiento de la red neuronal. Por lo tanto, es fácil de implementar, ocupa pocos datos de memoria, es intuitivo, requiere pocos ajustes en los hiperparámetros y combina las mejores características de AdaGrad y RMSProp [16].
- La propagación de raíz cuadrática media RMSprop se emplea para grandes conjuntos de datos. En esencia, realiza dos funciones, la primera labor es mantener un promedio del cuadrado de los gradientes, luego procede con una división entre el gradiente por la raíz del promedio para estimar la varianza [17].

2 METODOLOGÍA

2.1 MÉTODO DE INVESTIGACIÓN

El presente trabajo considera una investigación de tipo hipotético deductivo ya que parte de un procedimiento para falsear o aseverar tales hipótesis y confrontarlas con la realidad [18]. Para el caso de la resolución de laberintos, se entrena al modelo de la red mediante el aprendizaje por refuerzo profundo bajo las arquitecturas secuenciales y recursivas LSTM con la finalidad de conseguir salir del laberinto en el menor tiempo posible, tal como se presenta en la Figura 2.1.

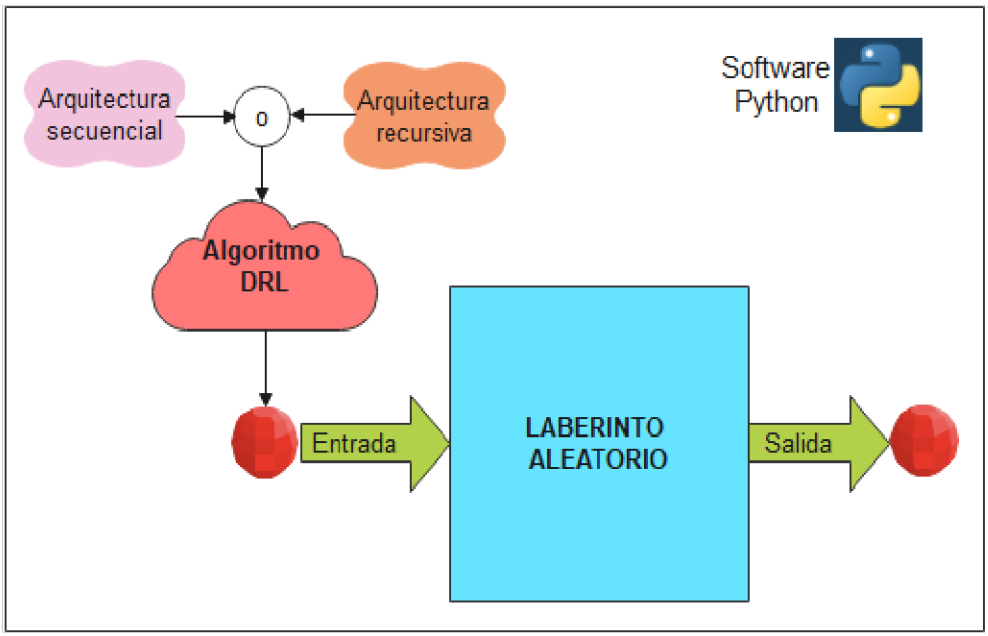


Figura 2.1. Esquema de resolución de laberintos

2.2 ENFOQUE MIXTO

Se considera un enfoque cualitativo ya que abarca la descripción específica de los eventos o interacciones observables del fenómeno de estudio, por lo que se adicionan los pensamientos, experiencias y/o creencias de los protagonistas de forma directa o indirecta. Además, el método cualitativo se caracteriza por ser inductivo ya que a partir de diferentes casos va induciendo resultados y validando gracias al empleo de la realidad empírica [18]. En cambio, el método cuantitativo se caracteriza por la medición numérica en base a la recolección de los datos y análisis estadístico, porcentual, entre otros; con el propósito de probar teorías y pautas de comportamiento [19].

Previo al uso de los algoritmos de aprendizaje por refuerzo profundo bajo las arquitecturas secuencial y recursiva, se detalla la definición y características de las librerías NumPy, PyTorch, Matplotlib, Tkinter y Jupyter.

2.2.1 NUMPY

Cuando se ocupa un alto volumen de datos y se requiere un análisis de estos y cálculo numérico, se puede recurrir a la librería NumPy de Python. Esta define una clase de objetos denominados arrays cuyo propósito es representar una estructura de datos organizada ya sea en forma de tabla o cuadrícula dependiendo del número de dimensiones 1D (eje 0), 2D (eje 0, eje 1) o 3D (eje 0, eje 1, eje 2) tal como se presenta en la Figura 2.2 [20].

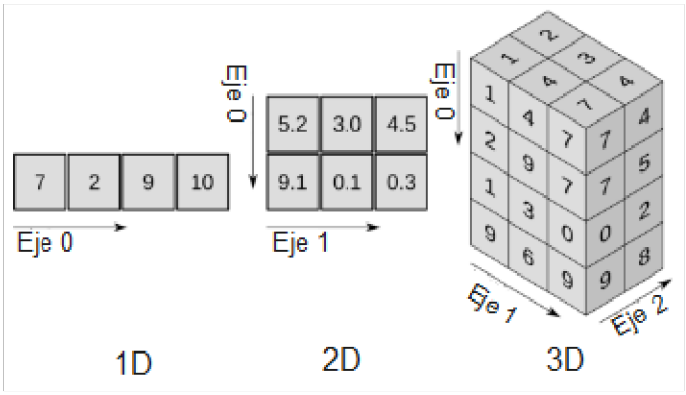


Figura 2.2. Dimensiones de un array

Por medio de la Tabla 2.1 se detallan algunas rutinas de la librería NumPy [21].

Tabla 2.1. Rutinas NumPy

Rutinas	Descripción
Numpy.random.rand ()	Forma una matriz con valores numéricos entre 0 a 1 de una distribución uniforme.
Numpy.argmax (a, axis, out)	Devuelve los índices de [21] los valores máximos, donde:
	a = es la matriz de entrada.
	axis = eje de matriz especificada, por tanto, es de tipo entero. En caso de que sea declare como “none”, indica que el índice se encuentra en la matriz plana.
	out = el resultado se coloca en una matriz si se declara la salida.
Numpy.zeros (shape, dtype, order, like)	Devuelve una nueva matriz de ceros, donde:
	shape = forma de la matriz de tipo entero y tupla de enteros.
	dtype = indica el tipo de datos, el valor por defecto es float64.
	order = almacena datos en orden de fila ‘C (predeterminada)’ o columna principal ‘F’ en la memoria.
	like = objeto que permite crear una matriz cuando no son de tipo NumPy.

Numpy.roll (a, shift, axis)	Los elementos ubicados en la última posición se reintroducen en la primera. La función del elemento “shift” es idéntica a “shape”.
Numpy.array (object, dtype, copy, order, subok, ndmin, like)	Crea una matriz, donde:
	object = es el objeto que expone la interfaz de la matriz.
	copy = crea una copia del objeto por defecto o si su valor es verdadero.
	subok = si el valor es verdadero, se pasan las subclases; caso contrario se convierte en una matriz de clase base (predeterminado).
	ndmin = indica el valor mínimo de dimensiones que posee la matriz resultante.

2.2.2 PYTORCH

PyTorch es una librería enfocada a las redes neuronales en Python, la cual se encarga de almacenar datos numéricos en una estructura multidimensional denominado tensor a fin de facilitar la ejecución de la unidad de procesamiento gráfico GPU y tratar los gradientes durante el aprendizaje profundo de la red [22]. El front-end de la librería PyTorch contiene varias clases y módulos con diferentes propósitos vistos en la Tabla 2.2 [23].

$$y = xA^T + B \tag{2.1}$$

Tabla 2.2. Componentes PyTorch

Bloques de construcción para gráficos		**Descripción**
Contenedores	Sequential	Contiene el modelo sequencial de una red neuronal.
	Module	Es una clase base de las redes neuronales.
Activación no lineal	nn.ReLU	Activa la función ReLU por elementos.
Capas recurrentes	nn.LSTM	Aplica una memoria LSTM multicapa a una secuencia de entrada.
Capas lineales	nn.Linear	Realiza una transformación lineal a los datos de entrada, expresado en la Ecuación 2.1.
Operaciones de creación	Zeros	Devuelve el valor numérico nulo al tensor.
	From_numpy	Se crea un tensor desde el array de numpy
Publicación por entregas	Load	Abre un objeto desde la dirección de un archivo

2.2.3 MATPLOTLIB

Matplotlib es un librería de Python que permite crear visualizaciones que contengan gráficos de análisis estadísticos, mapas de color, matrices, campos, parcelas, coordenadas no estructuradas, entre otras tal como se presenta en la Figura 2.3 [24].

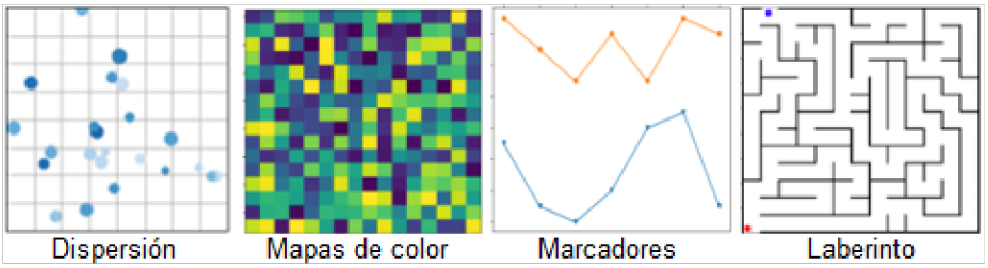


Figura 2.3. Gráficos interactivos

Por consiguiente, en la Tabla 2.3 se detallan algunos módulos y clases de la librería Matplotlib [24].

Tabla 2.3. Matplotlib

Módulos y clases	Descripción
Matplotlib.figure	Es una clase, la cual contiene todos los elementos de la trama o figura.
Matplotlib.backends.backend_tkagg	Renderiza la figura.
Matplotlib.pyplot.gcf ()	Se obtiene la cifra actual.
Matplotlib.pyplot.clf ()	Realiza una limpieza de la cifra actual.
Matplotlib.pyplot.plot ()	Realiza la gráfica en base a los parámetros (x, y).
Matplotlib.pyplot.savefig ()	Guarda la figura en un archivo con extensión png, pdf, svg, etc.
Matplotlib.pyplot.show ()	Muestra todas las cifras abiertas

2.2.4 TKINTER

Tkinter es un paquete implementado en el lenguaje C capaz de crear widgets individuales o interfaz de usuario como un objeto de Python. Además, cada widget posee las siguientes características [25]:

- No son creados de forma automática, sino que se implementan acorde a la línea de comandos utilizados.

- El botón y etiquetas son jerárquicos, es decir, están dentro de un marco y a su vez dentro de la ventana principal, tal como se presenta en la Figura 2.4.

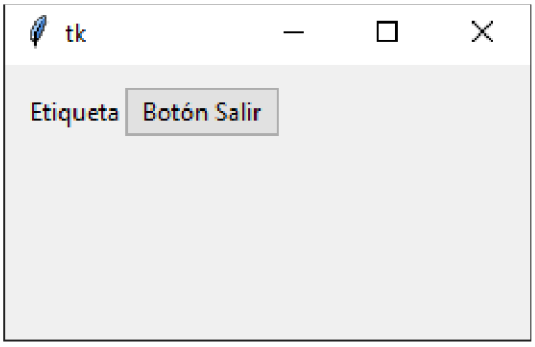


Figura 2.4. Tkinter

- Tiene opción de configuración tanto en la apariencia como en su comportamiento en los textos del botón y/o etiquetas.
- Finalmente, el programa se puede actualizar mediante el bucle de eventos Tk ().mainloop ().

2.2.5 JUPYTER

Jupyter es un software de acceso abierto, el cual posee diferentes paquetes orientadas a las siguientes aplicaciones [26]:

- Existe una interfaz de usuario JupyterLab potente y flexible gracias al uso de los componentes básicos como editor de texto, códigos, datos, exploradores, etc. Además, permite organizar y configurar flujos de trabajo en el campo de la informática científica, aprendizaje automático, periodismo computacional y ciencia de datos.
- Posee un programa kernel capaz de manejar solicitudes a fin de brindar una respuesta.
- Presenta un entorno interactivo basado en web denominado Jupyter Notebook para crear documentos con extensión JSON. Además, pueden emplear código, texto, gráficos, operaciones y modelos de extensión ipynb.
- Contiene un servidor multiusuario JupyterHub cuyo propósito es la administración, atención y representación de varios Notebooks individuales.

2.3 TIPO DE TRABAJO EXPLORATORIO

El tipo de trabajo exploratorio realiza un análisis de datos cualitativo seguidamente de otro cuantitativo. Existen dos modalidades de diseño, derivativo y comparativo; la primera utiliza

los resultados de los parámetros cuantitativos sobre una base del análisis cualitativo, teniendo como resultado final la comparación e integración de resultados bajo el enfoque mixto. Por tanto, se considera el diseño exploratorio bajo las siguientes tres etapas [19]:

- Conseguir temas, categorías u otros segmentos que puedan ser respaldados para llevar a cabo el análisis cualitativo.
- En base a la etapa anterior, construir un instrumento cuantitativo a fin de generar reactivos por cada tema o categoría.
- Validar el instrumento en base a la muestra probabilística.

El tipo comparativo consiste en la recolección y análisis de datos cualitativos y cuantitativos por separado, para luego ser comparados entre sí y formar un solo modelo. Por lo que resulta útil para un objeto de estudio y expandir sus resultados [19]. El presente trabajo utiliza como técnica de recolección de información el método de observación ya que permite conocer el comportamiento del fenómeno de estudio de forma directa. Por tanto, emplea una combinación de las dos modalidades, la primera utiliza el algoritmo DRL sobre cualquier laberinto y en base a reglas debe salir lo antes posible. Por consiguiente, se utiliza la modalidad comparativa cuando se trabaja con las arquitecturas secuencial y recursiva, a fin de observar cuál de las dos es mejor.

2.3.1 DISEÑO DE APRENDIZAJE DRL

En la Figura 2.5 se define el modelo de aprendizaje por refuerzo profundo para la resolución de laberintos, este a su vez se compone de la siguiente manera:

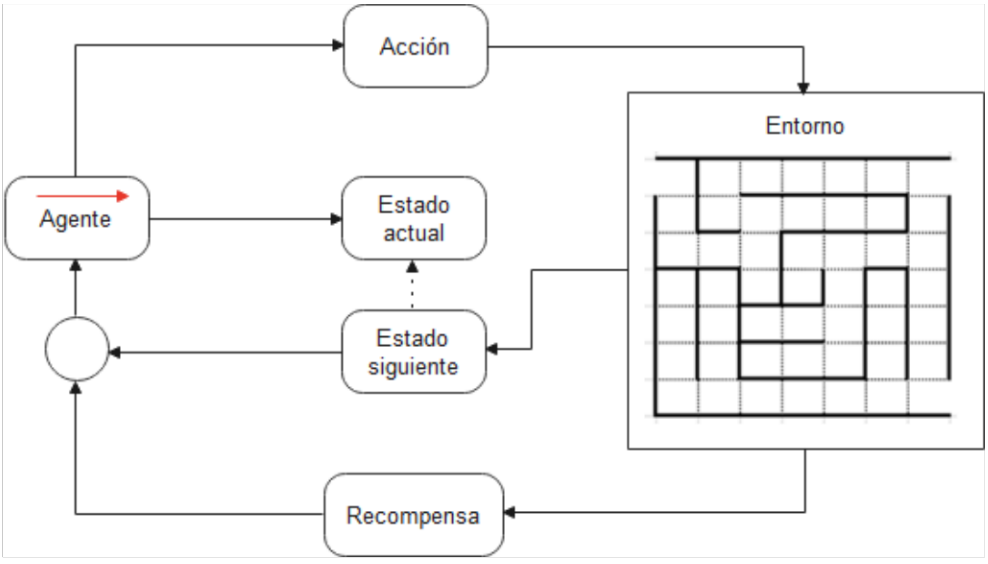


Figura 2.5. Modelo de diseño DRL

- El agente es el robot que recorre el laberinto con la finalidad de lograr salir en el menor tiempo posible.
- La acción se refiere a la decisión que toma el agente para desplazarse dentro del laberinto, por lo que se va a tener como referencia la posición actual.
- Una vez que el agente esté dentro del laberinto con alguna acción ya seleccionada, se obtiene los estados próximos, los cuales son enviados al agente para evaluar la situación respectiva.
- Se evalúa la movilidad del agente en base a la función de recompensa, es decir, si logra salir del laberinto obtiene 400 puntos o si descubre un camino que no ha recorrido anteriormente se gana unos puntos a favor. Caso contrario se reduce ante varias situaciones, tales como: cuando el agente se pierde en el laberinto y algún choque con alguna pared presentando valores negativos de 200 y 400 puntos; respectivamente.

2.3.2 DISEÑOS DEL LABERINTO Y AGENTE

La clase del laberinto se desarrolla en el software Python, el cual utiliza cinco funciones designadas como inicio, gráfica del laberinto, gráfica de la posición, acción y opciones indicado en la Figura 2.6.

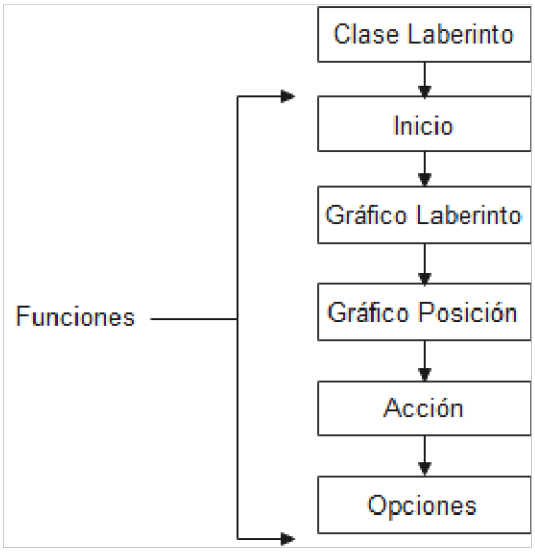


Figura 2.6. Diseño del laberinto y agente

2.3.2.1 Inicio

La función “Inicio” define la construcción del camino principal y demás trayectorias del laberinto, tal como se presenta en la Figura 2.7.

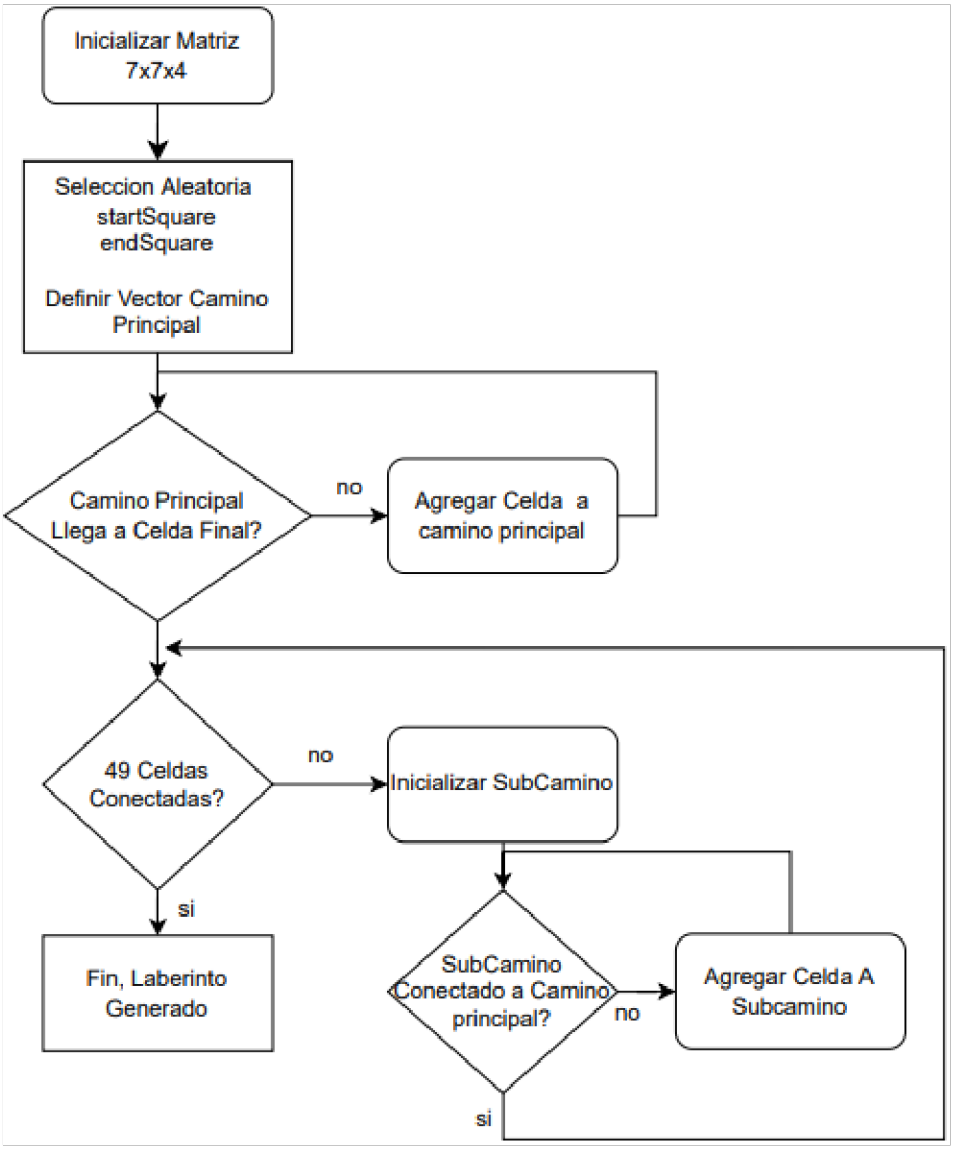


Figura 2.7. Laberinto generado

A continuación, se detalla una serie de pasos para el desarrollo de este:

- **Paso 1.** En la Figura 2.8 se observa los límites del laberinto representados como una matriz tridimensional 7x7x4, es decir posee 196 posiciones.

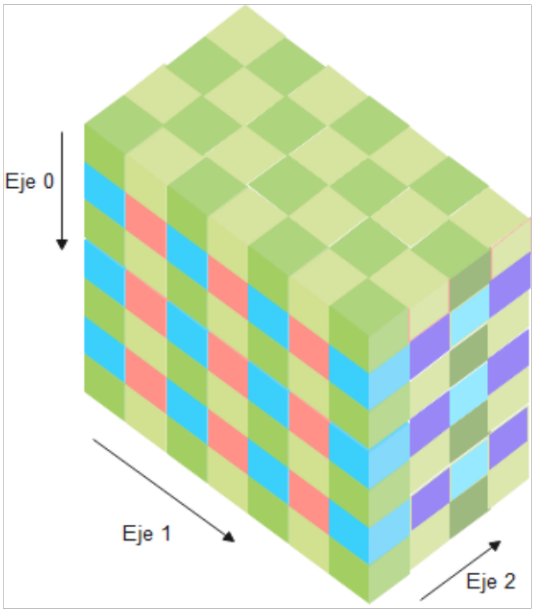


Figura 2.8. Esquema del laberinto (7x7x4)

- **Paso 2.** La entrada y salida del laberinto se diseña para que ingrese por el lado izquierdo y salga por el derecho, estos parámetros se designan de forma aleatoria en el rango de 0 a 6. Además, se plantea el laberinto como una base bidimensional, tal como se presenta en la Figura 2.12. Por consiguiente, se establece una entrada con el lado izquierdo abierto; de forma similar ocurre en la salida, pero con el lado derecho abierto.

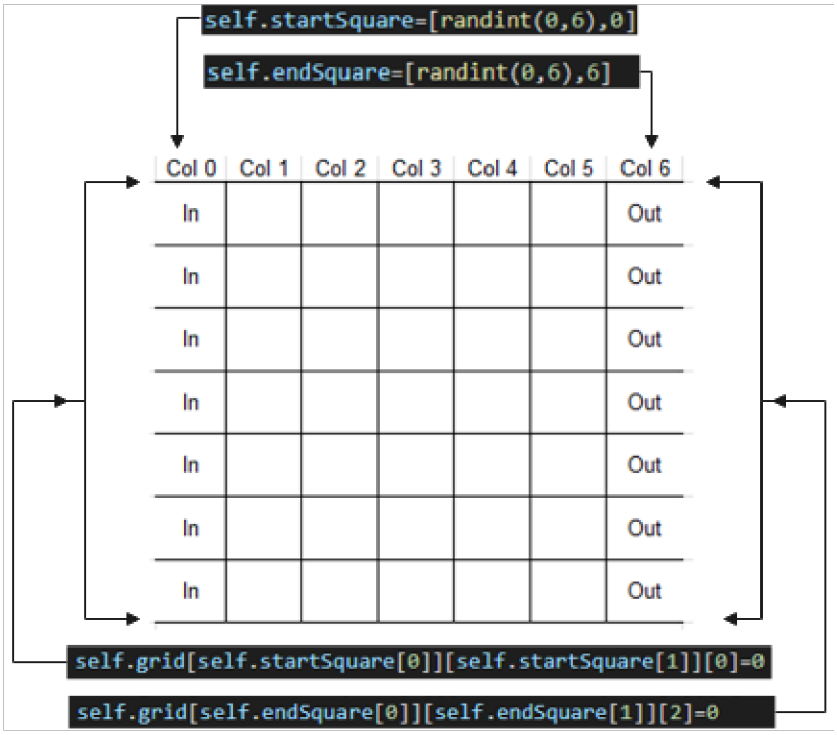


Figura 2.9. Entradas y salidas posibles del laberinto

- **Paso 3.** Cada pared del laberinto se representa como 1L (uno lógico) cuando existe una pared. Caso contrario, es 0L (cero lógico) ante uno o más paredes abiertas, tal como se visualiza en los ejemplos de la Figura 2.10.

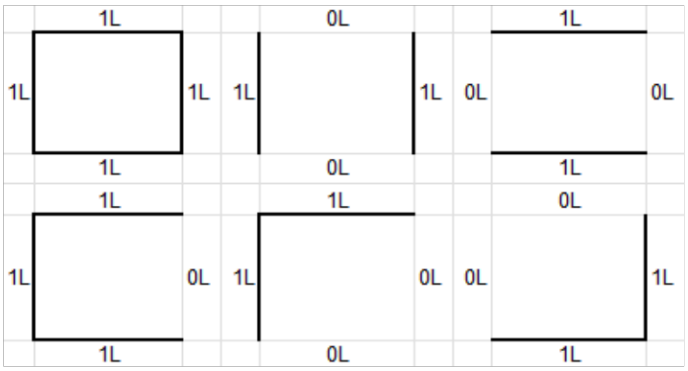


Figura 2.10. Caminos del laberinto

- **Paso 4.** Por lo tanto, cada posición se representa como a un vector de la forma $[\mathrm{L}, \mathrm{D}, \mathrm{R}, \mathrm{U}] = [\text{atrás}, \text{abajo}, \text{adelante}, \text{arriba}] = [0, 1, 2, 3]$, visto en la Figura 2.11.

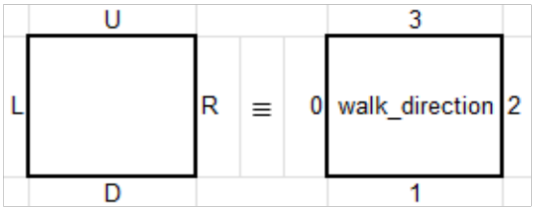


Figura 2.11. Vector posición

- **Paso 5.** Se establece el camino principal aleatorio en base a las posiciones indicadas en el paso 3, siempre y cuando la probabilidad sea menor a 0.4. Además, se define condiciones para evitar un regreso hacia la entrada o hacia el camino principal, tal como se expresa en los ejemplos de la Figura 2.12.

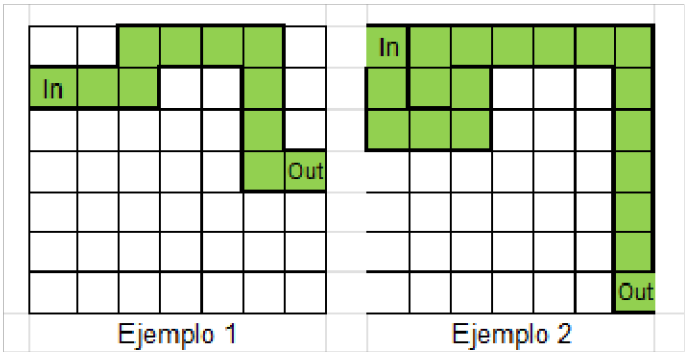


Figura 2.12. Caminos aleatorios principales del laberinto

- **Paso 6.** Finalmente, se va formando el resto de los caminos del laberinto hasta completar toda la cuadrícula, tal como se indica en la Figura 2.13.

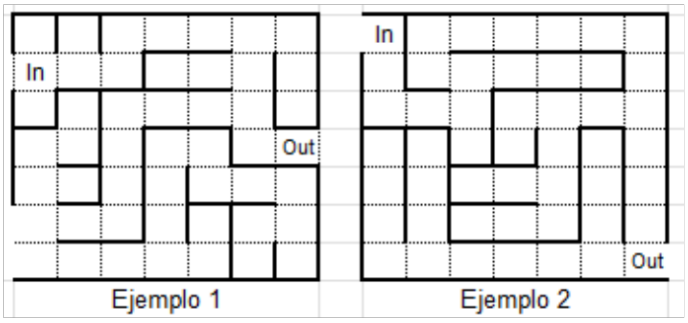


Figura 2.13. Diseño del laberinto

2.3.2.2 Gráficas del laberinto y posición del robot

Por medio de la librería Matplotlib se representan el agente (robot) y laberinto con sus respectivas unidades de medida, dada en píxeles. En la Figura 2.14 se observa un área de 175 píxeles^2, eso implica que cada posición contiene 25 píxeles^2.

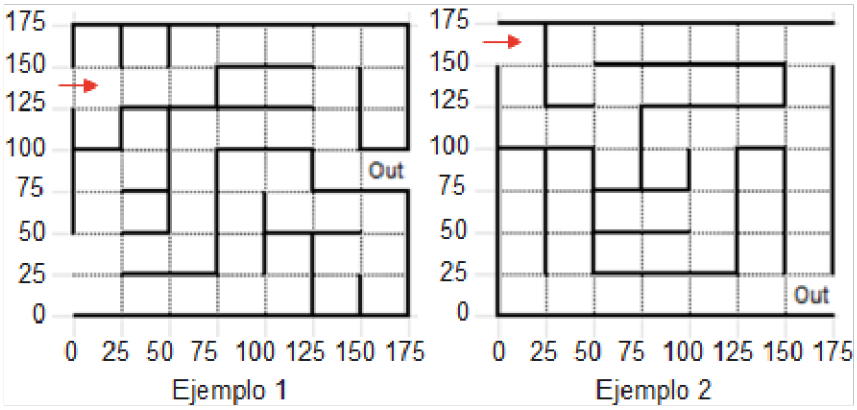


Figura 2.14. Diseño del laberinto

2.3.2.3 Acción y opciones

Se establece varias condiciones dentro de la función acción que va a tener el agente durante el desplazamiento dentro del laberinto indicados en la Figura 2.15, tales como un choque cuando insiste en seguir avanzando aun teniendo pared por delante o se regresa al punto de inicio. Otro caso, puede darse cuando el agente logre salir del laberinto, eso implica una gran cantidad puntos positivos ganados.

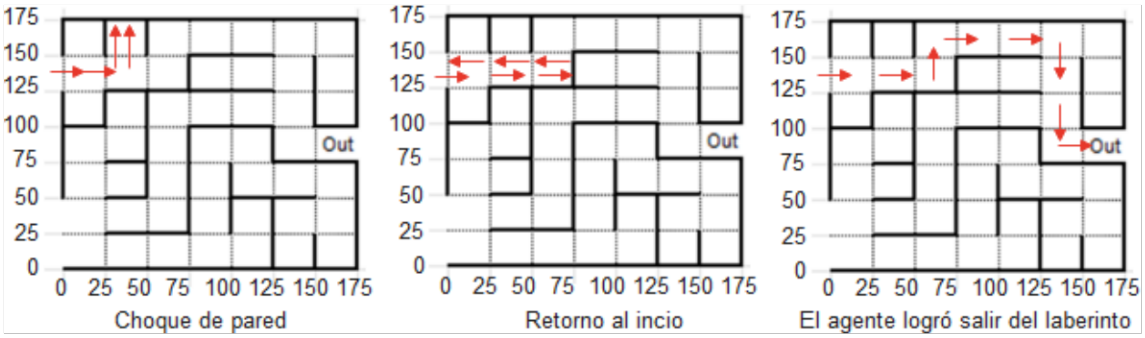


Figura 2.15. Diseño del laberinto

Por último, se tiene la función denominada opciones, el cual realiza una consulta al agente sobre que paredes existen al momento en cierta posición.

2.3.3 DISEÑO DE LA RED NEURONAL PROFUNDA

A través de la Figura 2.16 se presenta una estructura sobre el diseño de la red neuronal profunda que se basa en la arquitectura del modelo ya sea secuencial o recursiva LSTM. Luego pasa por un ciclo de entrenamiento para el desarrollo del algoritmo Q-learning; por consiguiente, se generan pruebas de la red y, por último, se realiza la repetición de la experiencia.

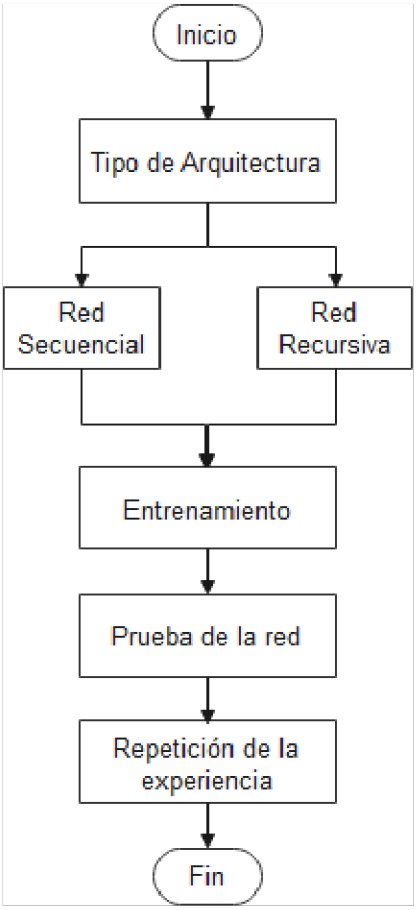


Figura 2.16. Diseño básico de la red neuronal profunda

2.3.4 ARQUITECTURA SECUENCIAL Y RECURSIVA

2.3.4.1 Configuración de la red secuencial

La arquitectura de la red secuencial se compone de varias capas, una de entrada L1, tres ocultas (L2, L3, L4) y una capa de salida L5, tal como se presenta en la Figura 2.17. La capa de salida posee un vector de 4 dimensiones ya que realiza cuatro posibles acciones de desplazamiento, tales como: arriba, abajo, izquierda y derecha.

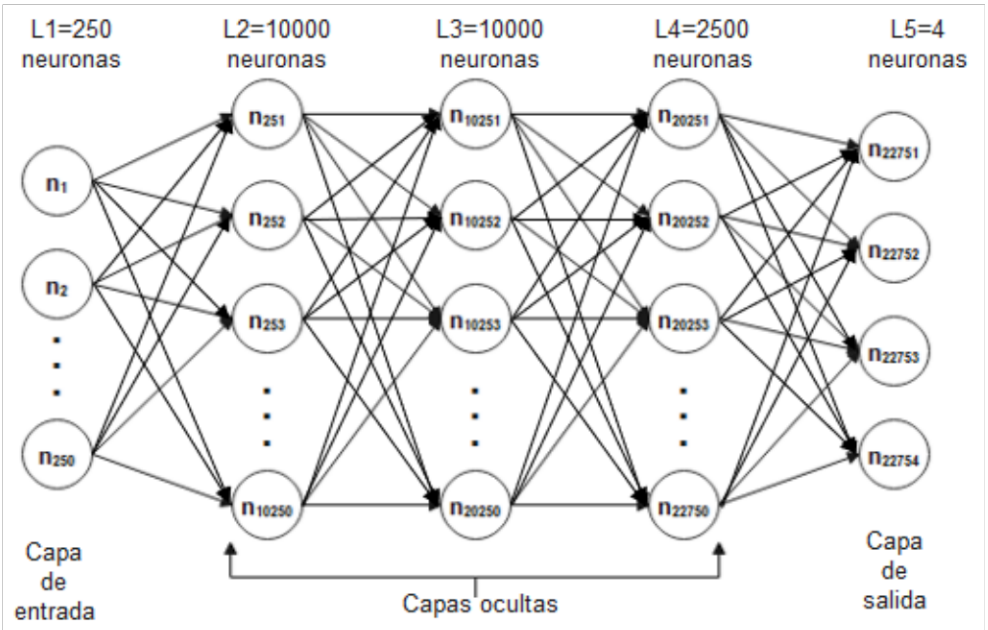


Figura 2.17. Estructura de la red neuronal secuencial

El modelo de la red secuencial se diseña mediante una transformación lineal entre un par de capas desde la entrada hacia la salida de la siguiente manera: (L1, L2), (L2, L3), (L3, L4) y (L4, L5). Luego, se designa funciones de activación ReLU hasta el penúltimo par de capas (L3, L4), tal como se presenta en la Figura 2.18.

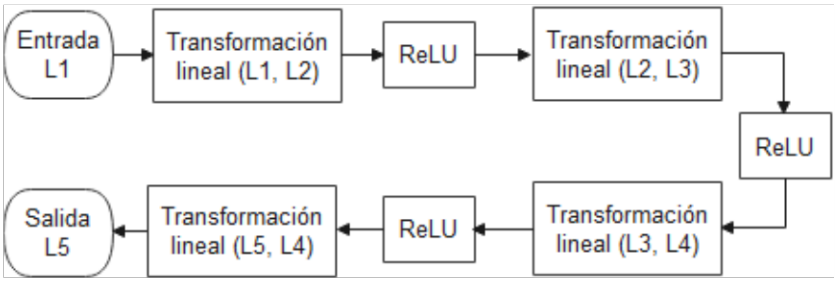


Figura 2.18. Modelo de la red secuencial

Por consiguiente, se define la función de pérdida de la librería Pytorch denominada MSELoss(), se establecen valores en los hiperparámetros: $\gamma = 0.8$ y $\propto = 0.0001$. Luego, se coloca una estrategia de probabilidad $\varepsilon = 0.8$ y se usa el optimizador Adam con el propósito de minimizar el error del modelo de la red con respecto a los datos.

2.3.4.2 Configuración de la red recursiva

La configuración de la red recursiva contiene memorias LSTM, la cual consta de una capa de entrada que cuenta con una neurona, 100 características en el estado oculto y 2 capas recursivas; esta última implica colocar un bloque LSTM seguido de otro. Por otro lado, se coloca una red secuencial, similar al Apartado anterior, salvo que se colocan 150 neuronas

en la capa de entrada y 13000 neuronas dentro de la capa oculta. Ambas redes se conectan en su totalidad con la capa de salida (4 neuronas), tal como se presenta en la Figura 2.19.

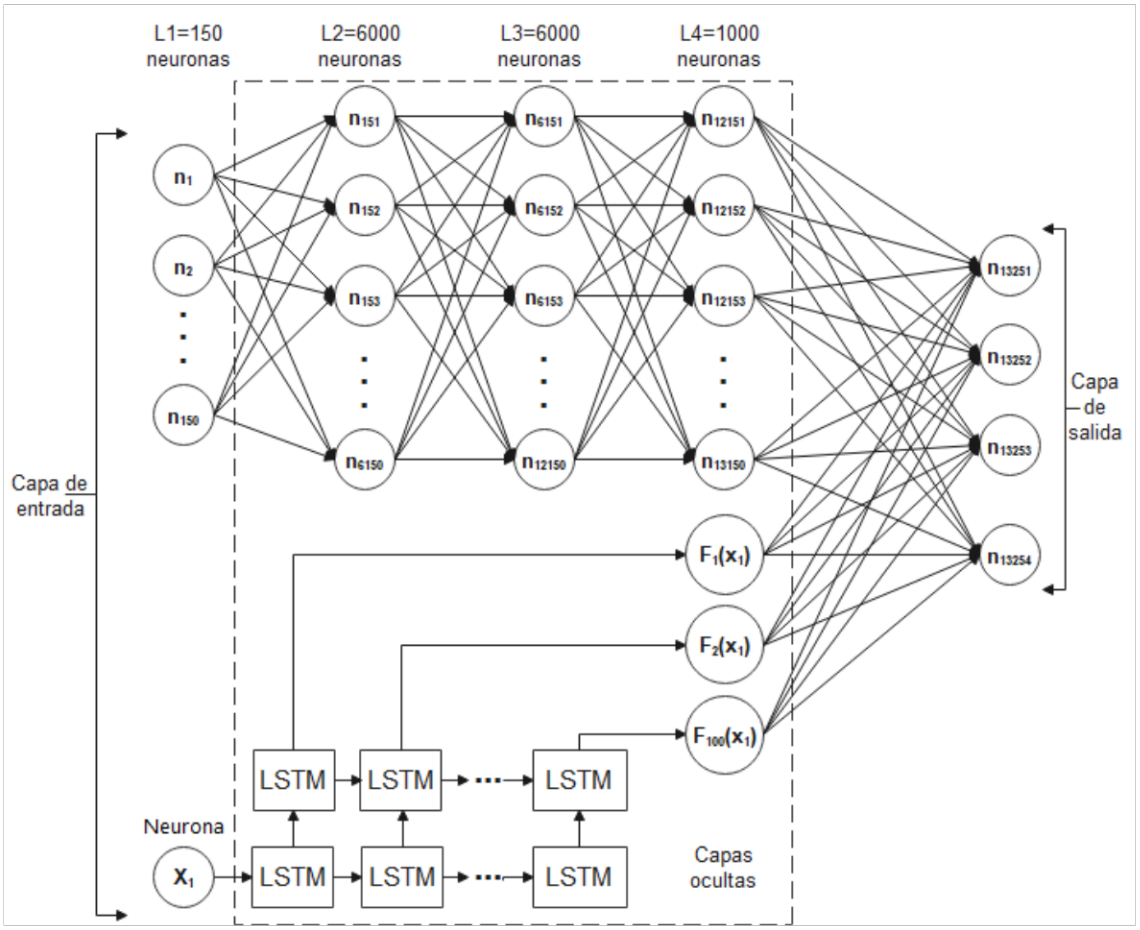


Figura 2.19. Estructura de la red neuronal recursiva LSTM

El modelo de la red emplea 4 transformaciones lineales, tal como se presenta en la Figura 2.20. Además, genera activaciones ReLU cuando se trabaja con la configuración secuencial (L1, L2), (L2, L3) y (L3, L4). En cambio, cuando se opta por ingresar la red recursiva, no es necesario colocar una activación ya que internamente la red cuenta con la función tangente hiperbólica. De tal manera que, cada entrada o salida posee los siguientes valores: L1 = 150, L2 = L3 = 6000, L4 = 1000, hidden_size = 100 y num_classes = 4.

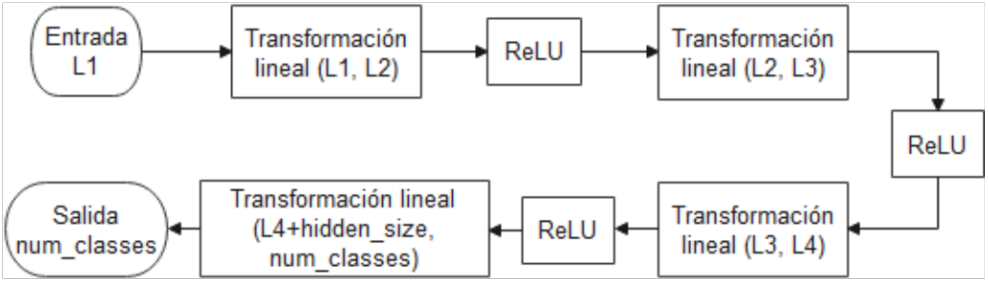


Figura 2.20. Modelo de la red LSTM

Otro punto importante del modelo de la red es definir los estados ocultos h_0 y de celda s_0 iniciales de cada elemento del lote. De modo que, devuelva un tensor relleno con la forma definida por $h_0 = c_0 = torch.zeros\ (*size, x.size(0), out)$, donde:

- $*size$ es el tamaño de la variable de argumentos, en este caso son las 2 capas recursivas.
- $x.size(0)$ cuenta el número de elementos a lo largo del Eje 0 del array.
- out representa el tensor de salida, es decir, son las 250 características del estado oculto.

Por consiguiente, se define la función de pérdida MSELoss(), se establecen valores en los hiperparámetros: $\gamma = 0.8$ y $\propto = 0.0002$. Luego, se coloca una estrategia de probabilidad $\varepsilon = 0.99$ y uso del optimizador Adam.

2.3.5 ENTRENAMIENTO Y PRUEBA DE LA RED

Ambas arquitecturas presentan un mismo procedimiento para el entrenamiento de la red, detallado en la Figura 2.21.

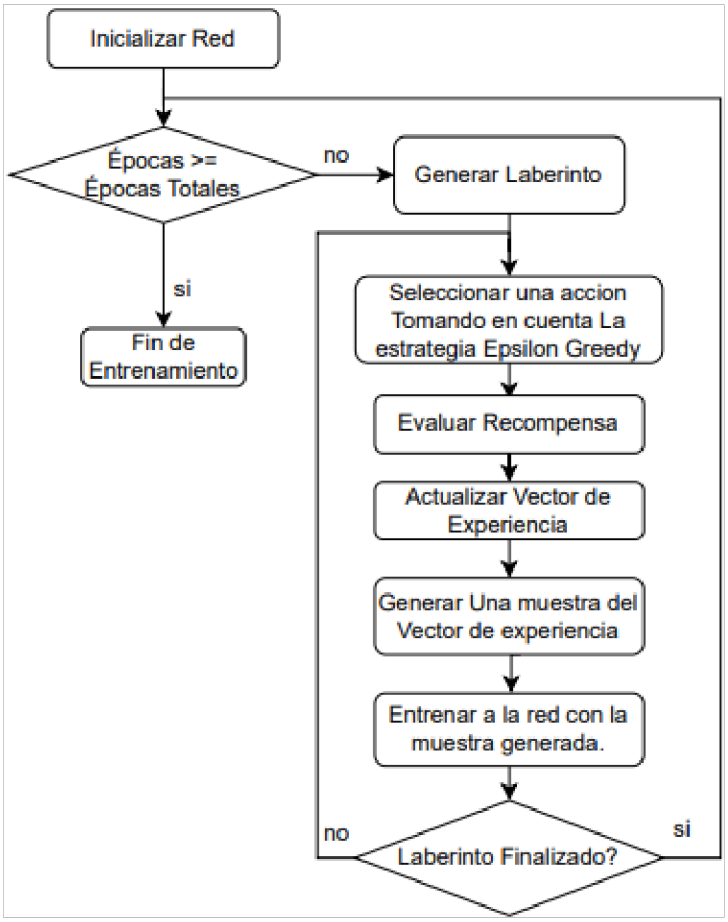


Figura 2.21. Entrenamiento de la red

A continuación, se describe una serie de pasos para el entrenamiento de la red:

- **Paso 1.** Se define un bucle de épocas y se almacena los valores de pérdida en una lista.
- **Paso 2.** Cuando las épocas son inferiores a las totales, se genera el laberinto y se toma una acción a, tomando en cuenta la estrategia Epsilon Greedy (método de probabilidad ε). Partiendo de 0.2 (1- ε), se escoge la acción que está asociado con el valor máximo de Q.

  Cabe mencionar, que por cada iteración ε va aumentando en 1 milésima de unidad.
- **Paso 3.** Se observa el nuevo estado S_{t+1} y se evalúa la recompensa R_{t+1}.
- **Paso 4.** Actualizar y generar la repetición de la experiencia (ver Apartado 2.3.6).
- **Paso 5.** Utilizando el nuevo estado, se ejecuta la red hacia adelante. Luego, almacenar el valor $max\big(Q(S_{t+1})\big)$.
- **Paso 6.** El objetivo de la red consiste en el entrenamiento de $\gamma * max\big(Q(S_{t+1})\big)$, ese valor se anula cuando se toma una acción y ya saló del laberinto, en ese caso el objetivo de la red es R_{t+1}.
- **Paso 7.** Dado que se desea entrenar o actualizar la salida con la acción tomada en el paso anterior, se aplica la Ecuación 1.1 para obtener el vector de salida.

$$Q(S_t, A_t) += \propto \left[R_{t+1} + \gamma * max\big(Q(S_{t+1}, a)\big) - Q(S_t, A_t)\right]$$

- **Paso 8.** Se procede a entrenar el modelo de la red y el proceso se realiza cíclicamente hasta completar las épocas totales y dar por finalizado el entrenamiento.

En esencia, la prueba de la red es idéntica al ciclo de entrenamiento, salvo que no se realiza ningún cálculo de pérdida y repetición de la experiencia. Además, tiene como propósito la obtención de las predicciones ejecutando la red hacia adelante.

2.3.6 REPETICIÓN DE LA EXPERIENCIA

Cuando se juega en laberintos aleatorios, el algoritmo no puede simplemente memorizar una secuencia de pasos para salir del laberinto, ya que las paredes de este nunca van a ser las mismas. Por tal motivo, se requiere de uno que sea capaz de entrenarse ante esta diversidad y recorrerlo hasta llegar a la meta en el menor tiempo posible. Ante ello, se

desarrolla una representación denominada repetición de la experiencia, esta consiste en brindar actualizaciones por lotes de un esquema de aprendizaje en línea (ver Figura 2.22), detallado a continuación:

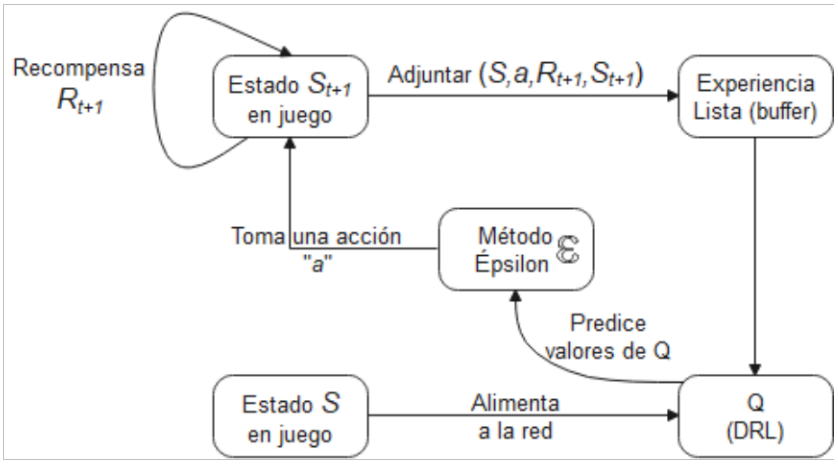


Figura 2.22. Repetición de la experiencia

- En el estado S tomar una acción a, con ello se observa un nuevo estado S_{t+1} junto con la recompensa R_{t+1}.
- Guardar los datos (S, a, S_{t+1}, R_{t+1}) en una lista.
- Cada experiencia se almacena en una lista hasta completarla.
- Si la lista de reproducción es mayor al tamaño del lote, comienza el entrenamiento del mini lote. Además, se guarda un modelo de archivo cuando el resto de una división entre la iteración y un valor numérico de 10 unidades es cero.
- En caso de llenarse esta memoria, se escoge un subconjunto de la lista de reproducción.
- Por consiguiente, descompone cada experiencia en tensores de mini lotes separados.
- Realizar la iteración del subconjunto, de manera que se calcula las actualizaciones de los valores $Q1$ del mini lote para obtener gradientes. Luego se vuelve a calcular $Q2$ con los siguientes estados, pero sin gradientes.
- Luego, se guarda en una matriz destino (Y) y se almacena S de cada memoria en X.

- Por último, se toma X y Y para formar un mini lote para el entrenamiento por lotes. En caso de que la matriz esté llena debido a las épocas anteriores, se sobrescribe los valores antiguos en la matriz de memoria de reproducción de la experiencia.

2.3.7 DISEÑO DE LA INTERFAZ

La interfaz conlleva algunos criterios nuevos y algoritmos ya presentados en este apartado, tales como:

- Se colocan las arquitecturas de las redes secuencial y recursiva indicadas en el Apartado 2.3.4; luego, se cargan los modelos ya entrenados.
- Se crea un laberinto aleatorio con el agente, exactamente igual a lo que se observó en el Apartado 2.3.2.2.
- Por consiguiente, se define en la Figura 2.23 una pantalla de la aplicación realizada en el paquete Tkinter, donde se ingresa al programa con la opción "Comenzar".

Figura 2.23. Ingreso al programa laberinto

- Una vez ingresada a la aplicación se realiza la prueba de la red, similar a lo mencionado en el Apartado 2.3.5. Luego, el programa (Figura 2.24) permite generar un nuevo laberinto bajo las arquitecturas secuencial o recursiva e indicar el avance o retroceso del agente.

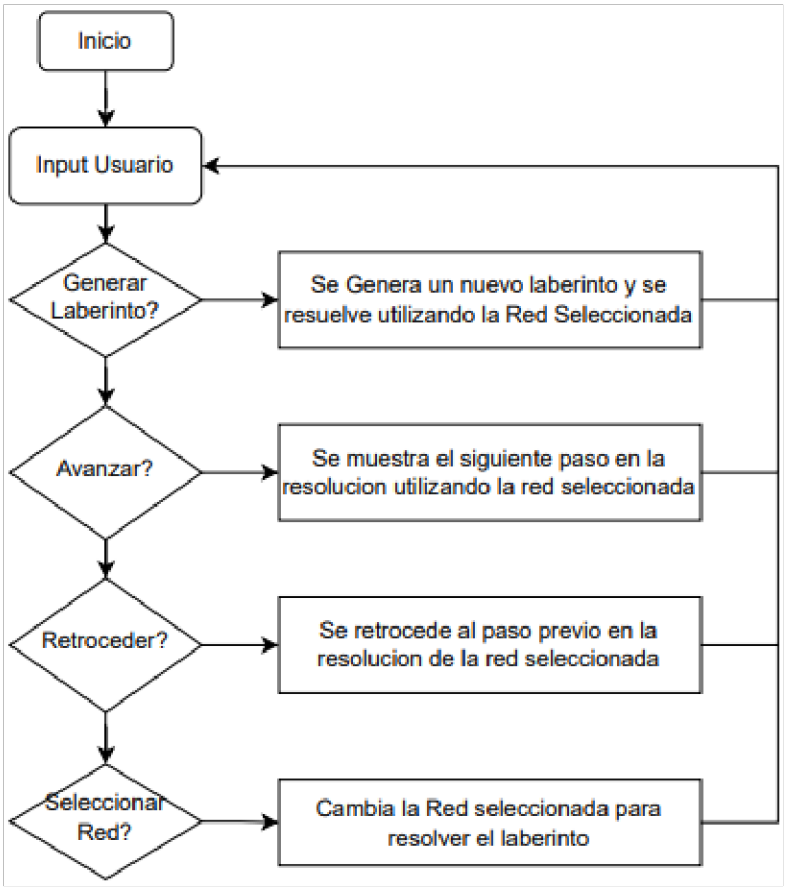


Figura 2.24. Diseño de la interfaz

Por ende, en la Figura 2.25 se observa una pantalla cuyo manejo es realizado por el usuario.

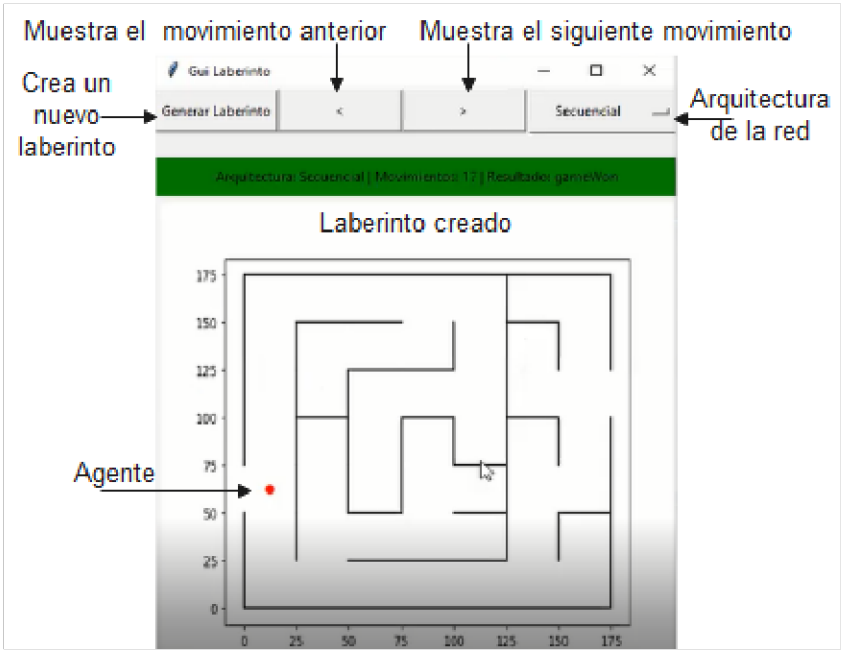


Figura 2.25. Pantalla principal de la interfaz

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 RESULTADOS

Esta sección engloba los resultados producto del diseño propuesto, es decir, pruebas con las redes neuronales entrenadas y funciones de pérdida, desarrollo de resolución del laberinto en base a las acciones tomadas por el agente, comparación entre las arquitecturas de tipo secuencial y recursiva e implementadas con la lógica Q-learning en el aprendizaje de refuerzo profundo.

3.1.1 RESULTADOS DEL ENTRENAMIENTO

Una vez entrenada la red bajo las arquitecturas secuenciales y recursivas DRL se toman varias muestras en base al número de laberintos generados, que tiene como propósito visualizar el porcentaje de partidas ganadas y el valor de la recompensa total. Por consiguiente, se realizan pruebas para los 50, 100, 150 y 200 laberintos aleatorios, tal como se indica en la Tabla 3.1 y Anexo I.

Tabla 3.1. Resultados de las pruebas realizadas

Laberintos generados		**Prueba 1**	**Prueba 2**	**Prueba 3**	**Prueba 4**
		50	100	150	200
Arquitectura secuencial	Partidas ganadas	28 (56%)	71 (71%)	97 (65%)	125 (63%)
	Recompensa total	6110	23493	21434	30919
Arquitectura recursiva	Partidas ganadas	17 (34%)	53 (53%)	64 (43%)	83 (42%)
	Recompensa total	-7652	3524	-9940	-13956

A través de la Figura 3.1, se destaca que la arquitectura secuencial es mejor a la recursiva con relación a las partidas ganadas, ya que tiene la capacidad de resolver laberintos entre un 56 a 71%; mientras que la red recursiva no supera el 53% de ganancias.

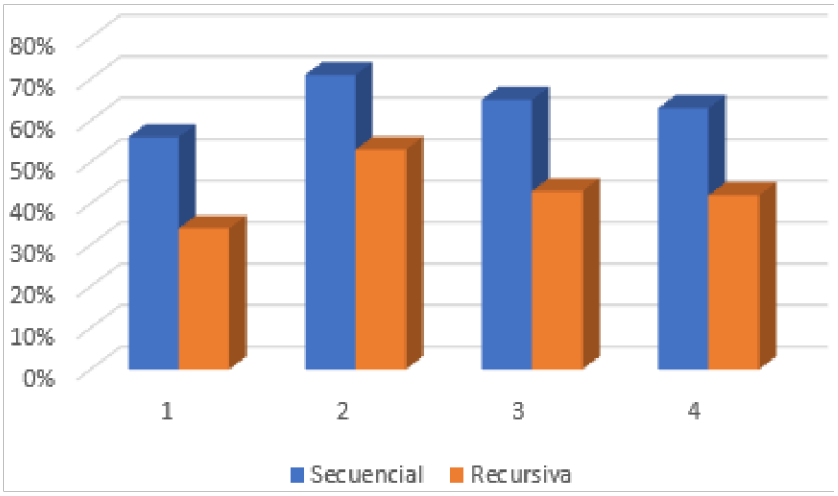


Figura 3.1. Muestreo en la resolución de laberintos

Así mismo, sucede con las recompensas totales de las dos redes mencionadas; tal es el caso de la red secuencial que posee más de 6000 puntos de las partidas ganadas. En cambio, con la red recursiva presentan recompensas negativas debido a que no supera el 50% de las partidas ganadas, tal como se visualiza en la Figura 3.2.

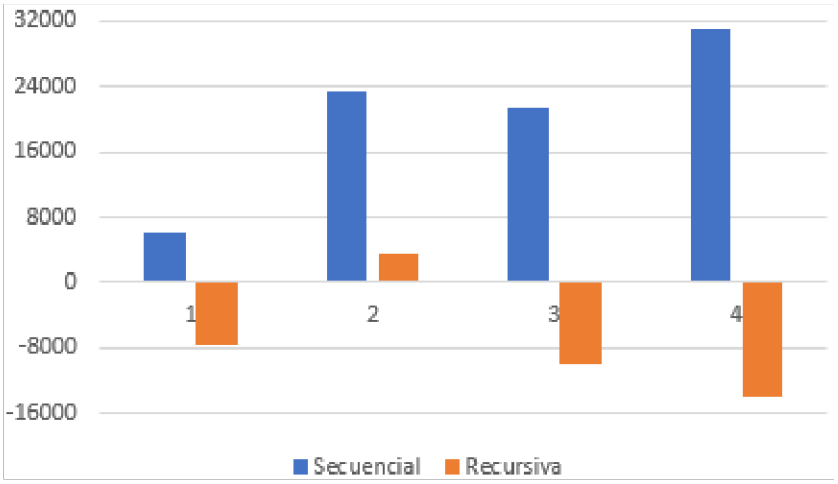


Figura 3.2. Recompensa total de las arquitecturas secuencial y recursiva

3.1.2 FUNCIONES DE PÉRDIDA

La función de perdida MSE (error cuadrático medio o global) es un factor importante en las redes neuronales, ya que permite evaluar la desviación entre los valores reales de las observaciones utilizadas durante el aprendizaje por refuerzo profundo y las predicciones realizadas, eso quiere decir que, ante una disminución de este valor, la red se vuelve más eficiente como es el caso de la red secuencial, visto en la Figura 3.3.

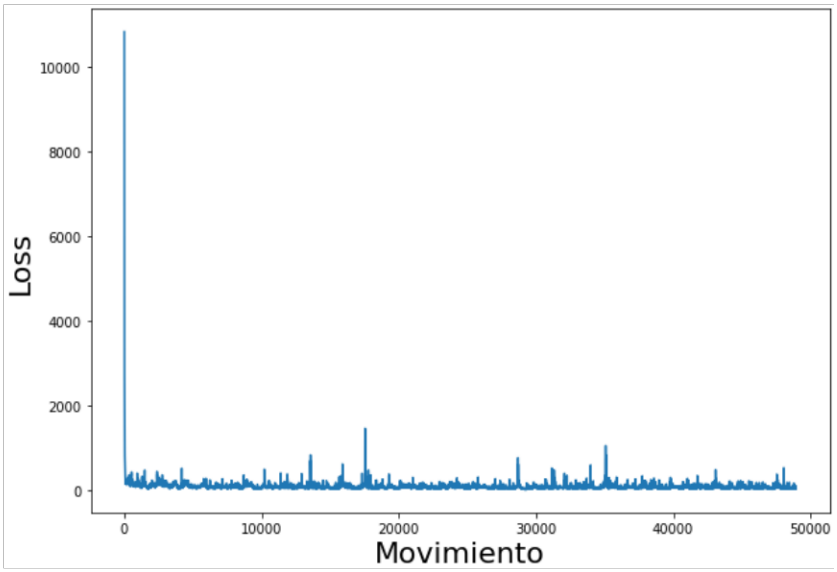


Figura 3.3. Resultados con las arquitecturas secuencial

Con respecto a la red recursiva, disminuye su valor de pérdida; sin embargo, presenta variaciones durante los movimientos desarrollados por cada partida del laberinto, siendo su valor superior al de la red secuencial, tal como se visualiza en la Figura 3.4.

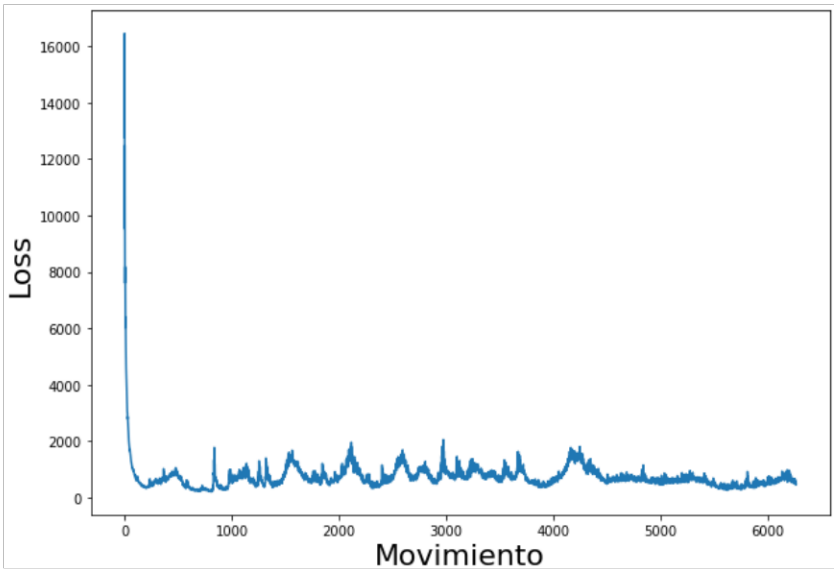


Figura 3.4. Resultados con las arquitecturas recursiva

3.1.3 COMPARACIÓN ENTRE ARQUITECTURAS

Por medio de las Figuras 3.5.a) y b) y Anexo II se observan como el agente resuelve el laberinto ante el uso de las arquitecturas secuenciales y recursivas; respectivamente. Cabe notar que ambas recorren caminos similares, pero con una cantidad de movimientos que pueden o no ser iguales entre estas.

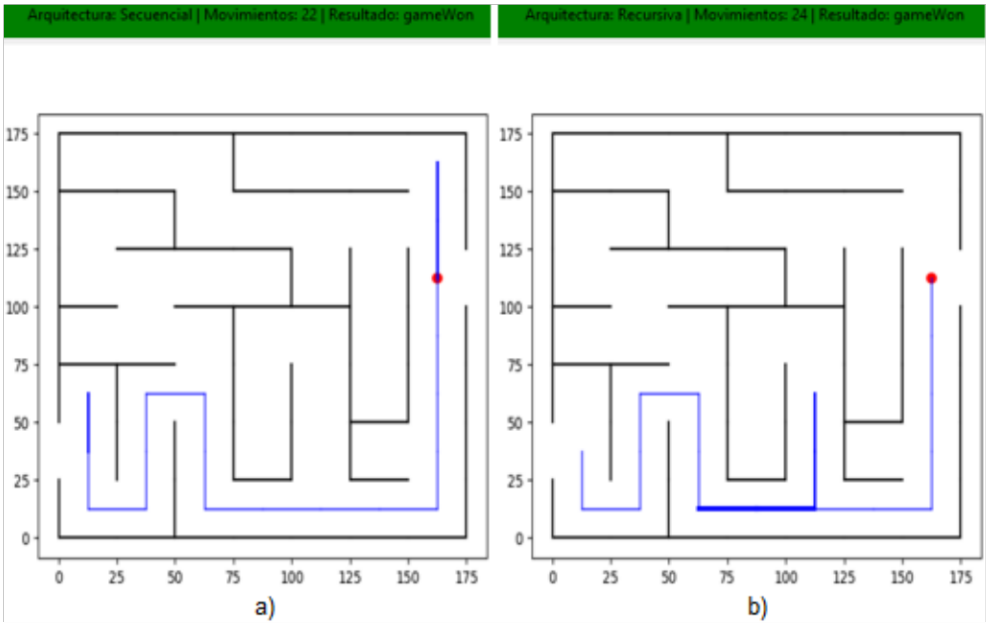


Figura 3.5. Resultados con las arquitecturas a) secuencial y b) recursiva

3.1.4 SITUACIONES DE RESOLUCIÓN EN EL LABERINTO

Acorde a las reglas del laberinto vistas en el Apartado 2.3.1, se tienen 4 situaciones que posee el agente, detalladas a continuación:

- Cuando el agente logra salir del laberinto, obtiene como resultado la ganancia de este, tal como se indica en la Figura 3.6.

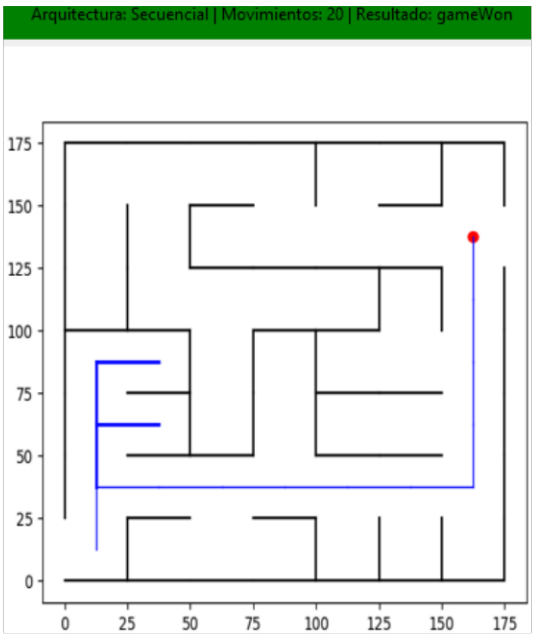


Figura 3.6. Ganancia cuando el agente sale del laberinto

- En caso de pérdida, puede darse cuando el agente se choca con alguna pared (ver Figura 3.7.b), no logra encontrar el camino (ver Figura 3.7.a) o si supera los 49 movimientos.

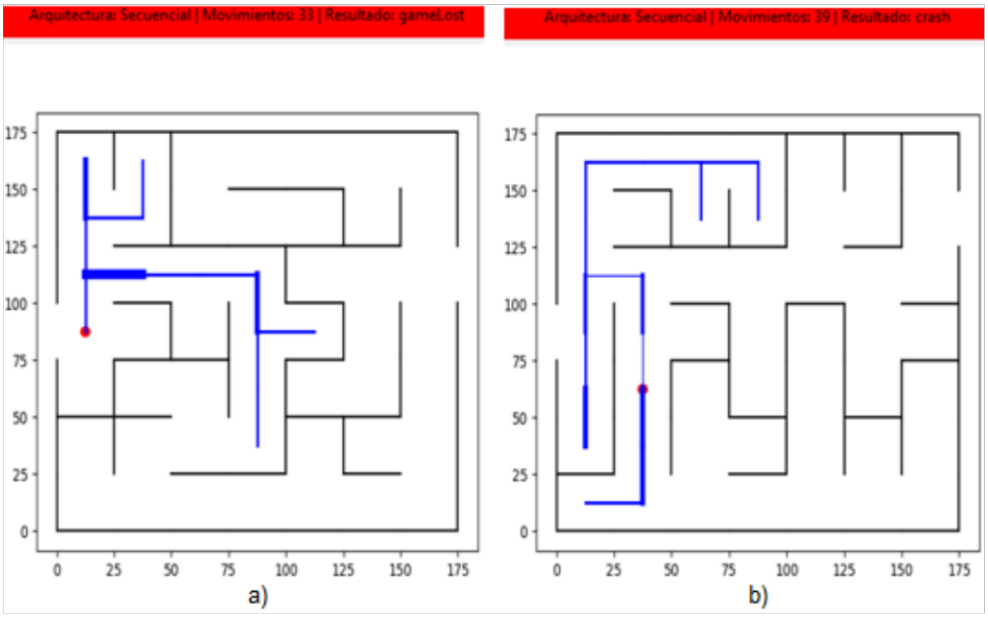


Figura 3.7. Puntos perdidos en el laberinto

3.1.5 CONTEO DE PASOS

Se conoce que cada laberinto generado de forma aleatoria posee un camino directo hacia la salida de este, tal como se indica en la Figura 3.8.a). Por tanto, al aplicar una arquitectura secuencial (ver Figura 3.8.b) o recursiva, el agente es capaz de resolver el laberinto sin tener que memorizar el camino, haciendo uso de predicciones, en este caso realiza 14 movimientos para salir del laberinto.

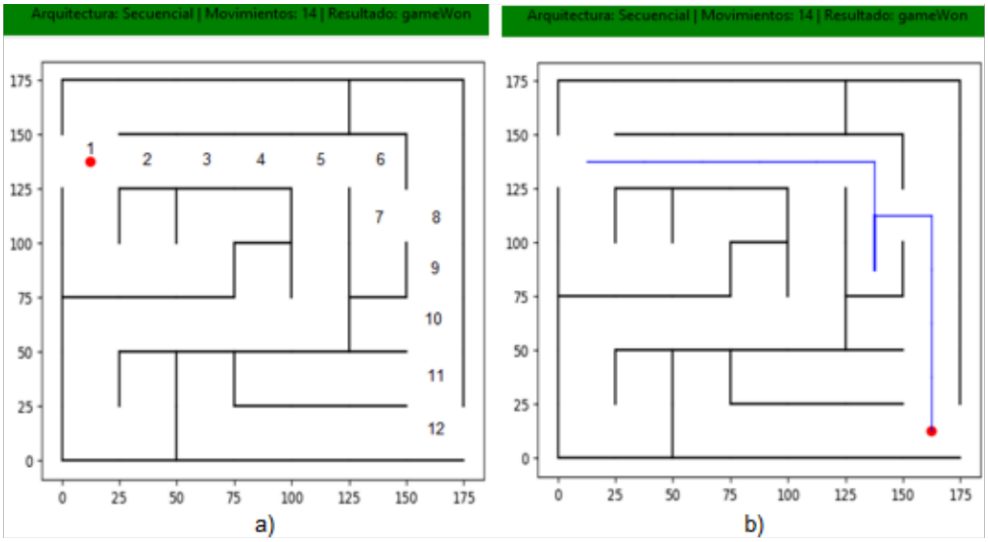


Figura 3.8. Resultados con las arquitecturas a) secuencial y b) recursiva

3.2 CONCLUSIONES

- Cada vez se ha ido extendiendo en los últimos años el tema de las redes neuronales, cuyas aplicaciones se orientan al uso doméstico como es el caso del robot Roomba, cumpliendo actividades de traslado de objetos en cuestión de minutos, entre otras. Además, existen diversos tipos de redes neuronales; sin embargo, en el presente trabajo de integración se realiza el análisis sobre la arquitectura secuencial, la cual consiste en conectar múltiples capas totalmente conectadas entre sí hasta completar la predicción. En cambio, la red recursiva conecta cada salida de la capa hacia las anteriores, por lo que se considera una red generalizada recurrente; en otras palabras, la red escoge la información a través del propio registro (salida) a fin de generar un nuevo resultado.

- El algoritmo de aprendizaje por refuerzo profundo define 5 factores fundamentales como el agente (robot) que es el encargado de recorrer y salir del laberinto en el menor tiempo posible, el entorno (laberinto), la acción o movimientos que realiza el agente, por tanto, se recoge datos sobre los estados actuales/posteriores y la recompensa. La arquitectura de las redes neuronales poseen capas de entrada, ocultas y salida, esta última corresponde a los cuatro movimientos (izquierda, derecha, arriba, abajo) que realiza el agente. Por consiguiente, la red secuencial presenta 250 y 22500 capas de entrada y ocultas; respectivamente. En cambio, la segunda arquitectura se diseña bajo modalidades secuenciales y recursivas LSTM, a tal punto que se configura con 151 neuronas para la capa de entrada y 22100 neuronas en las capas ocultas junto con 2 capas LSTM.

- La función de recompensa se determina en base a los objetivos a corto, en este caso se proporcionan puntuaciones positivas cuando el agente descubre un camino que no ha recorrido anteriormente. En cambio, los objetivos a largo plazo se dan ante acciones positivas cuando el agente logra salir del laberinto y negativas (penalizaciones) ante un choque con alguna pared, se pierde o no logra salir del laberinto.

- El laberinto es una matriz 3D que contiene 196 posiciones (7x7x4), donde cada pared representa 1L (uno lógico), caso contrario es un camino abierto. Además, posee condiciones de diseño, tales como: la entrada y salida se define a la izquierda y derecha; respectivamente, otro factor importante es que tanto la entrada como la salida se designa de forma aleatoria. Por consiguiente, se establece un camino directo aleatorio, de tal manera que se va formando el resto de los caminos hasta

completar la cuadrícula de dimensión cuadrada 7x7. Ante ello, empleando el algoritmo Q-learning bajo las arquitecturas mencionadas anteriormente se realiza el entrenamiento de la red.

- La resolución de laberintos se realiza a través del paquete Tkinter y la librería Matplotlib, los cuales generan visualizaciones de los componentes de la red DRL. Además, permite seleccionar el tipo de arquitectura (secuencial o recursiva), define laberintos aleatorios y botones donde se puede observar los movimientos del agente. En este medio, se logra constatar que la red secuencial (56% - 71%) presenta mejores características de entrenamiento que la recursiva (34% - 53%).

3.3 RECOMENDACIONES

- Las pruebas fueron realizadas en un computador Intel Core i7 con 16 GB de RAM instalada, por lo que el tiempo de entrenamiento de la red secuencial se realiza en 8 días para 4000 épocas. Por tanto, si se desea volver a entrenar a la red se recomienda utilizar como mínimo un procesador de esta característica.

- El uso de algoritmos secuencial y recursivo DRL se adaptan en laberintos cuyos caminos son cuadrados (90 grados). Sin embargo, se puede plantear como trabajo a futuro el rendimiento de la red basado en otros laberintos ya sea de tipo unicursales medievales donde los ángulos de los caminos son cuadrados, circulares y poligonales y en laberintos multicursales donde existen más de un solo camino para llegar a la meta.

- Como trabajo a futuro se recomienda emplear otros métodos de aprendizaje para el entrenamiento de la red ya sea mediante el algoritmo Police network o redes adversarias cuyo propósito sea la resolución de laberintos.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] A. Sastre, "Xiaomi Cyberdog: así es el robot guardián para aterrorizar a los intrusos", Zone, Grupo ADSL, 2021. [En línea]. Available: https://cincodias.elpais.com/cincodias/2021/08/11/lifestyle/1628695377_962388.html. [Último acceso: 14 septiembre 2021].

[2] C. iRobot, "Robot aspirador Roomba", iRobot, [En línea]. Available: https://www.irobot.es/roomba. [Último acceso: 25 septiembre 2021].

[3] Infobae, "Los robots de Amazon que han reducido el trabajo de un día a menos de una hora", Infobae, 2019. [En línea]. Available: https://www.infobae.com/america/tecno/2019/02/20/los-robots-de-amazon-que-han-reducido-el-trabajo-de-un-dia-a-menos-de-una-hora/. [Último acceso: 25 septiembre 2021].

[4] M. Morales, "Grokking Deep Reinforcement Learning", New York, NY, USA: Manning Publications Co, 2020, pp. 3-37.

[5] R. Sutton y A. Barto, "Reinforcement Learning: An Introduction", 2nd ed., London, England: The MIT Press, 2020, pp. 1-8.

[6] CER, Reglamento para la categoría “Robot Laberinto”, 2019. [En línea]. Available: https://cer.cedia.edu.ec/dmdocuments/CER/2019/Reglamentos/ReglamentoRobotLaberinto.pdf. [Último acceso: 16 noviembre 2019].

[7] F. Chollet, "Deep Learning with Python", New York, NY, USA: Manning Publications Co, 2018, pp. 1-13.

[8] F. Alonso, "Redes Neuronales y Deep Learning. Capítulo 1: Preludio", Future Space, 2021. [En línea]. Available: https://www.futurespace.es/redes-neuronales-y-deep-learning-capitulo-1-preludio/. [Último acceso: 18 noviembre 2021].

[9] A. Zai y B. Brown, "Deep Reinforcement Learning in Action", New York: Manning Publications Co, 2020, pp. 55-83.

[10] Bosch, A.; Casas, J.; Lozano, T., "Deep Learning y Fundamentos", 1st ed., Barcelona, España: Reverté-Aguilar, 2019, pp. 47-207.

[11] L. Denoyer y P. Gallinari, *"Deep Sequential Neural Network",* Sorbonne Universites UPMC, 2014.

[12] Guo, Q.; Yu, Z.; Wu, Y.; Liang, D.; Qin, H.; Yan, J., *"Dynamic Recursive Neural Network",* Computer Vision Foundation, 2019, pp. 5147-5156.

[13] S. Kostadinov, "Recurrent Neural Networks with Python Quick Start Guide", Packt Publishing, 2018.

[14] G. Mayanglambam, "Optimizadores de aprendizaje profundo", Towards Data Science, 2020. [En línea]. Available: https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f. [Último acceso: 17 enero 2022].

[15] M. Zeiler, «"Adadelta: Un método de Tasa de Aprendizaje Adaptativo",» ArXiv, 2012.

[16] J. Brownlee, "Introducción suave al algoritmo de optimización de Adam para el aprendizaje profundo", Machine Learning Mastery, 2021. [En línea]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Último acceso: 17 enero 2022].

[17] V. Bushaev, Comprender RMSprop: aprendizaje más rápido de redes neuronales, Towards Data Science, 2018. [En línea]. Available: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a. [Último acceso: 17 enero 2022].

[18] M. Reyes, "Metodología de la Investigación", Ciudad de México: Secretaría de Educación Pública, 2016.

[19] C. Fernández y P. Baptista, "Metodología de la Investigación", México D.F.: McGraw Hill, 2014.

[20] Aprendeconalf, "Programación con Python", Wowchemy Website Builder, 2020. [En línea]. Available: https://aprendeconalf.es/docencia/python/manual/. [Último acceso: 23 diciembre 2021].

[21] NumPy, «"NumPy v1.21 Manual",» Sphinx, 2021. [En línea]. Available: https://numpy.org/doc/stable/#. [Último acceso: 23 diciembre 2021].

[22] J. Torres, "Introducción al aprendizaje por refuerzo profundo: Teoría y práctica en Python", Barcelona, España: Independently Published, 2021.

[23] PyTorch, "Pytorch de la investigación a la producción",» PyTorch, [En línea]. Available: https://pytorch.org/. [Último acceso: 22 diciembre 2021].

[24] Matplotlib, "Matplotlib: visualización con Python",» The Matplotlib Development team, 2021. [En línea]. Available: https://matplotlib.org/. [Último acceso: 28 diciembre 2021].

[25] Python, "Tkinter — Python interface to Tcl/Tk", Python Software Foundation, 2021. [En línea]. Available: https://docs.python.org/3/library/tkinter.html?highlight=tkinter#module-tkinter. [Último acceso: 28 diciembre 2021].

[26] Jupyter, "Jupyter Project Documentation", The Jupyter Trademark, 2021. [En línea]. Available: https://docs.jupyter.org/en/latest/. [Último acceso: 28 diciembre 2021].

5 ANEXOS

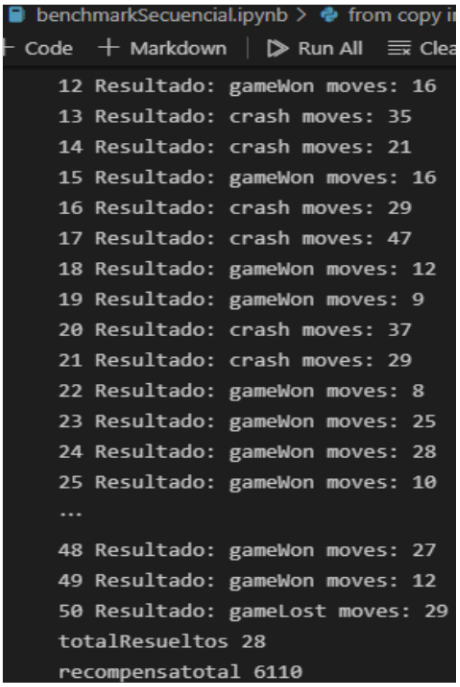
ANEXO I. Efectividad de las arquitecturas secuencial y recursiva DRL

ANEXO II. Comparación entre las arquitecturas secuencial y recursiva DRL

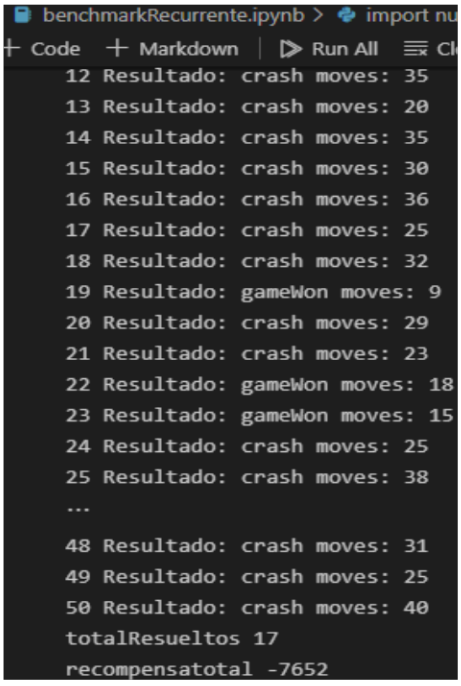
ANEXO I

En el Anexo I se observa las pruebas realizadas de los laberintos generados en el Apartado 3.1.1.

- 50 laberintos generados.

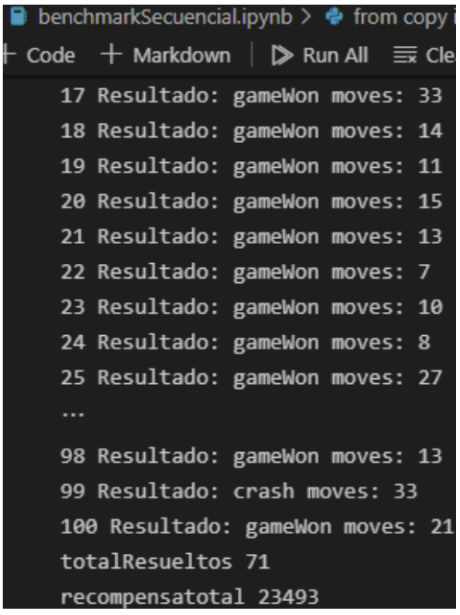


Arquitectura Secuencial

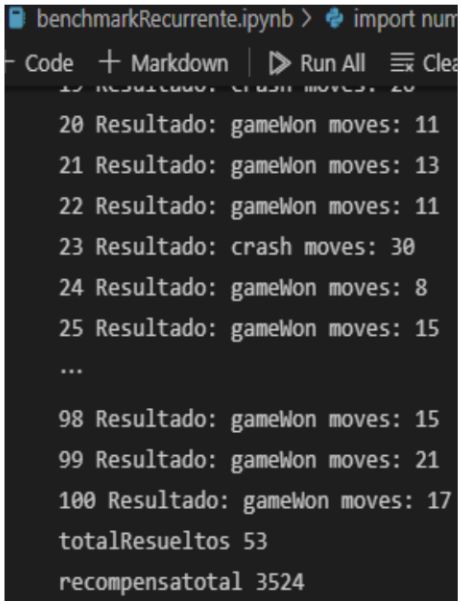


Arquitectura Recursiva

- 100 laberintos generados.

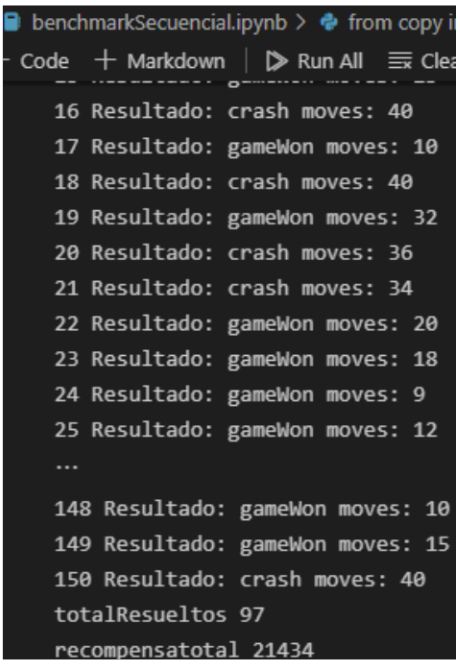


Arquitectura Secuencial

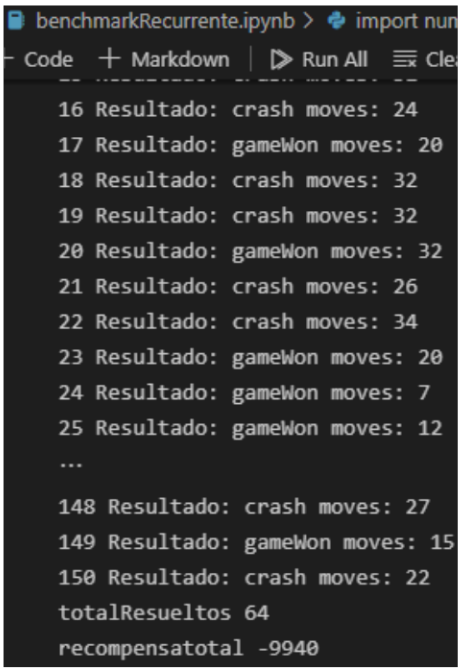


Arquitectura Recursiva

- 150 laberintos generados.

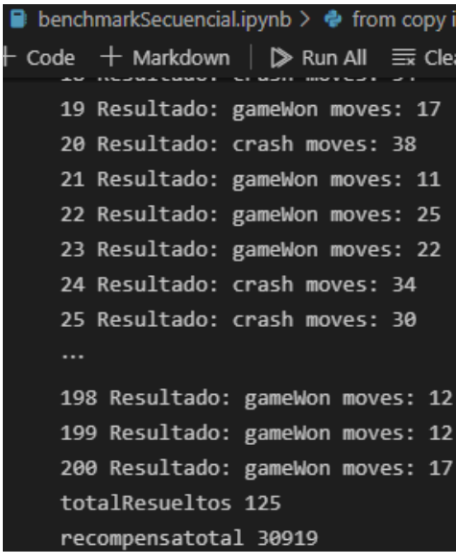


Arquitectura Secuencial

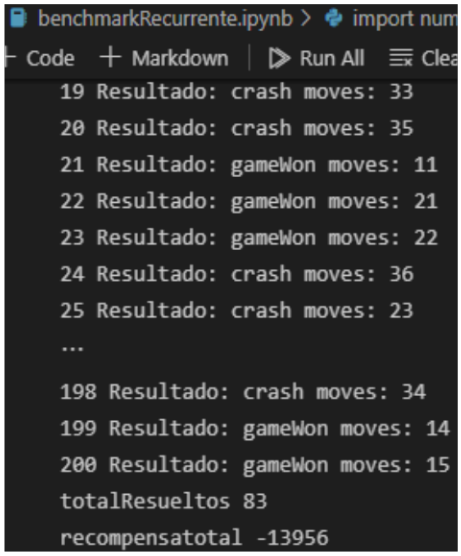


Arquitectura Recursiva

- 200 laberintos generados.



Arquitectura Secuencial

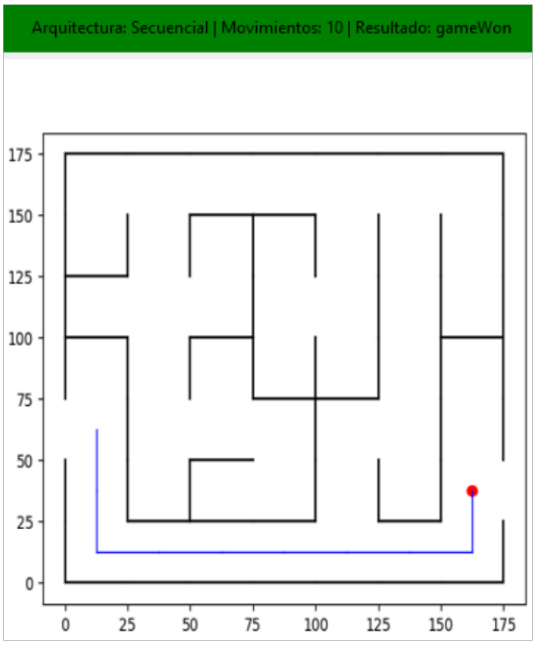


Arquitectura Recursiva

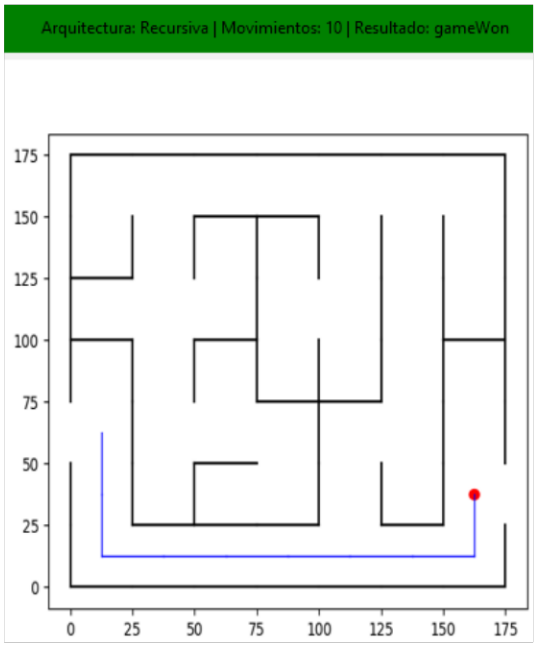
ANEXO II

En el Anexo II se detallan comparaciones entre las arquitectura secuencial y recursiva en la resolución del laberinto.

- Prueba 1.

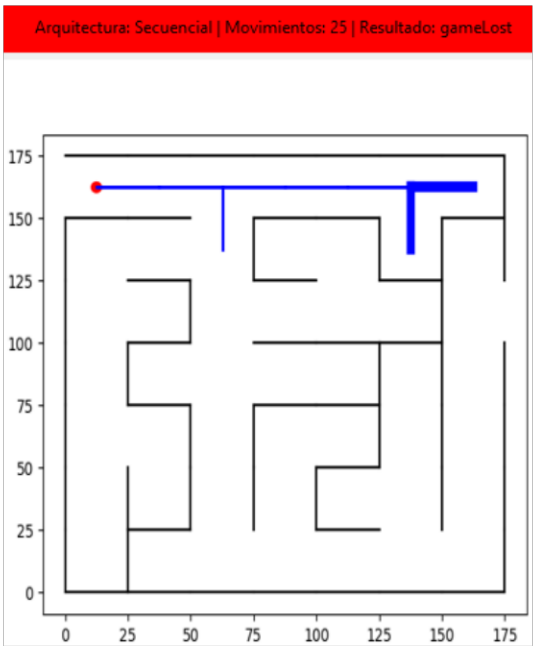


Arquitectura Secuencial

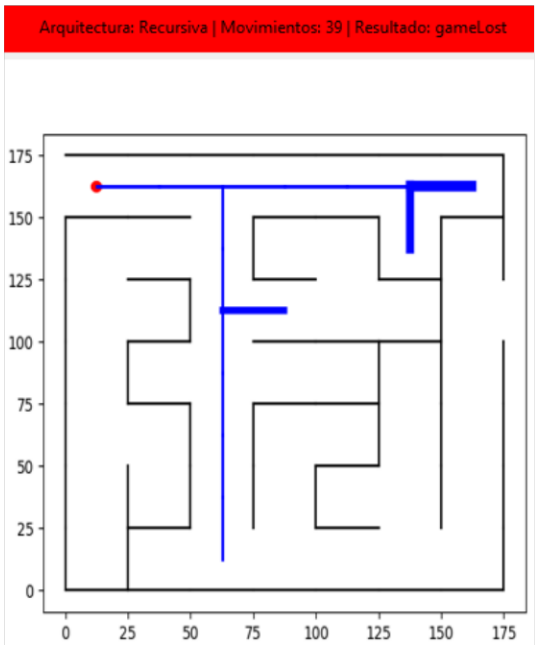


Arquitectura Recursiva

- Prueba 2.

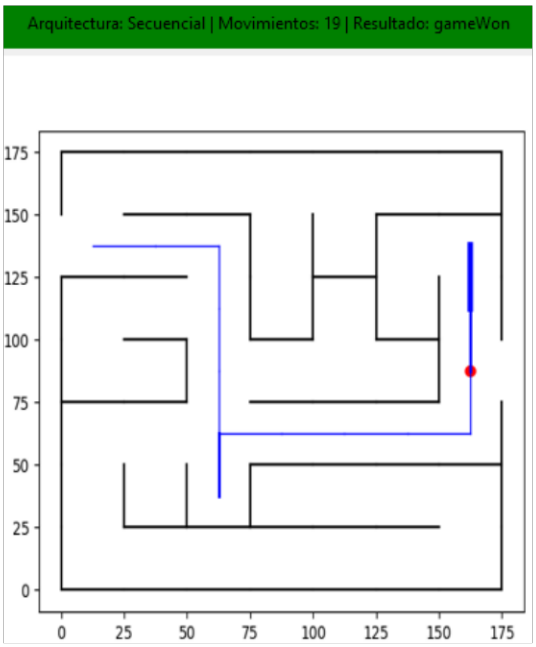


Arquitectura Secuencial

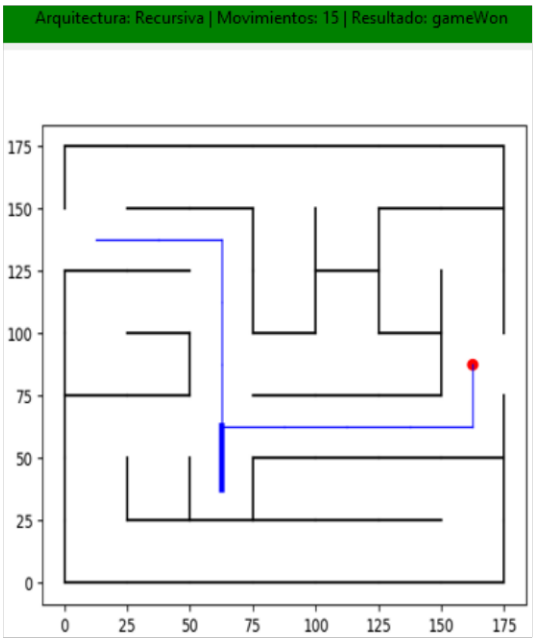


Arquitectura Recursiva

- Prueba 3.



Arquitectura Secuencial



Arquitectura Recursiva