UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

REINFORCEMENT LEARNING PARA PROBLEMA DE PLANIFICACIÓN FORESTAL CON INCERTIDUMBRE EN PRECIOS Y DEMANDAS

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN GESTIÓN DE OPERACIONES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INDUSTRIAL

FABIÁN IGNACIO SEPÚLVEDA LILLO

PROFESOR GUÍA:
Andrés Weintraub Pohorille

PROFESOR CO-GUÍA:
Jaime Carrasco Barra

COMISIÓN:
Juan Velasquez Silva

Este trabajo ha sido parcialmente financiado por:
Fondecyt 1220893/2023

SANTIAGO DE CHILE
2024

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE:
MAGÍSTER EN GESTIÓN DE OPERACIONES
MEMORIA PARA OPTAR AL TITULO DE:
INGENIERO CIVIL INDUSTRIAL
POR: FABIÁN IGNACIO SEPÚLVEDA LILLO
FECHA: 2024
PROF. GUÍA: ANDRÉS WEINTRAUB POHORILLE
PROFESOR CO-GUÍA: JAIME CARRASCO BARRA

REINFORCEMENT LEARNING PARA PROBLEMA DE PLANIFICACIÓN FORESTAL CON INCERTIDUMBRE EN PRECIOS Y DEMANDAS

Esta tesis aborda la aplicación del Aprendizaje por Refuerzo (Reinforcement Learning, RL) en la planificación de la cosecha forestal bajo condiciones de incertidumbre, un problema estocástico de alta dimensionalidad. Se centra en la variante de programación estocástica multietapa para la cosecha de celdas en un área forestal, caracterizada por decisiones secuenciales adaptativas frente a incertidumbres como fluctuaciones en los precios de la madera y la demanda del mercado, modeladas mediante un árbol de escenarios.

El trabajo se enfoca en implementar y analizar técnicas avanzadas de RL, comparándolas con métodos tradicionales de optimización bajo incertidumbre. Se utilizan algoritmos como Deep Q-Networks (DQN), Dueling DQN, Double Dueling DQN y Proximal Policy Optimization (PPO), adaptándolos al contexto específico de la planificación forestal. Se busca evaluar la efectividad, eficiencia computacional y escalabilidad de estas técnicas en el sector, así como su robustez ante diferentes escenarios estocásticos.

El enfoque metodológico incluye una revisión exhaustiva de la literatura en RL y su aplicación en problemas de alta dimensionalidad, una formulación detallada del problema forestal adaptado para RL y el diseño e implementación de los algoritmos mencionados. Se realizan experimentos y análisis para probar la viabilidad de RL en este contexto, destacando la capacidad de estas técnicas para adaptarse a la variabilidad y la incertidumbre inherentes a la planificación forestal.

Los hallazgos subrayan la complejidad de aplicar RL en ambientes estocásticos, resaltando la necesidad de un diseño cuidadoso de la función de recompensa, el equilibrio entre exploración y explotación, y la precisión en la estimación de funciones de valor. Futuras investigaciones deberían enfocarse en mejorar la función de recompensa, refinar la búsqueda de hiperparámetros y explorar nuevas estructuras de redes neuronales para abordar la dinámica estocástica del problema.

Only the disciplined ones are free in life.
If you aren't disciplined,
you are a slave to your moods.
You are a slave to your passions.
That's a fact
E.K

Agradecimientos

Primero que todo, quiero expresar mi más sincera gratitud a mi familia, cuyo apoyo incondicional ha sido esencial en cada una de las decisiones que he tomado a lo largo de los años. Mis padres, Jorge y Cecilia, y en especial a mi hermano Simón, han sido pilares fundamentales en mi vida. A Simón, le deseo que disfrute su experiencia en la Facultad tanto como yo lo hice y que siga avanzando con la misma dedicación y pasión.

Agradezco profundamente al Profesor Andrés Weintraub por su paciencia, comprensión y apoyo durante el desarrollo de este trabajo. Su pasión es una fuente inagotable de inspiración y ha sido un privilegio tenerlo como guía durante este tiempo. También extiendo mi agradecimiento a Lucas Murray por su invaluable apoyo y orientación.

No puedo dejar de lado a las amistades forjadas durante mi paso por la Facultad, partiendo por ese gran grupo formado en plan común, la proyi: PL - Gracias por el eterno apoyo con tu rigor matemático -, Poaboom, Max, Santiago, Manteca (DaIAP), Feña, Leiva, Franco, Pérez, Millanao, Corvalán, Zapata, Carlitos, Jorge, Méndez, el tarde y otros más. Quedan grandes anécdotas, las cuales al mirar en retrospectiva solo me hacen reír a carcajadas y sentir lo bueno que fue el camino por la Escuela. Asimismo, agradezco a los Sedex, mis amigos de Geo: Severino, Alexis, Ore, Tomi, Loco Yáñez, BC, Maxi, quienes hicieron de esa pequeña gran vuelta en la Geología algo inolvidable.

A mis amigos de la vida por siempre estar ahí: Axel, Cuevas, Ruiz, Marita, Nacha, Josefa, Micky y otros que van y vienen. Espero que sigamos cultivando la amistad como lo hemos hecho a lo largo de todos estos años.

Además, agradezco a Marcos, años de amistad - incluyendo un pequeño interludio - para luego invitarme a formar parte del proyecto en el cual trabajamos hoy en día junto a Lucas. Les agradezco la confianza y libertad entregada para poder desarrollar mi carrera en la dirección que siempre estuve buscando. También, mi gratitud al equipo de DTW por su apoyo constante: Warlee, Camilo, Belén, Lucho, Nicolito, Leo, Chaima y todos los demás que se han incorporado en este último tiempo.

Finalmente, agradezco a las diferentes bandas que me han acompañado por tantos años, tanto en vivo como en mis audífonos, han hecho que la vida sea más amena: Porcupine Tree, King Crimson, Pearl Jam, Pink Floyd, Megadeth, Powderfinger, Alice in Chains, Steven Wilson, Gustavo, Charly, Los Tres, Buckley, The Black Keys, RHCP, Faith No More, Paul Gilbert, U2, Soundgarden, todos los proyectos de Cornell y muchas otras más.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación del estudio

La emergencia de la inteligencia artificial ha proporcionado herramientas y técnicas poderosas para resolver problemas complejos en diversos campos de estudio. En particular, el Reinforcement Learning (RL) ha demostrado ser un paradigma crucial para el diseño de algoritmos que optimizan la toma de decisiones bajo incertidumbre. Sin embargo, el uso de RL en problemas de alta dimensionalidad representa un área de investigación en constante desarrollo. La alta dimensionalidad conlleva un vasto espacio de estados o acciones, lo que puede dificultar o, en ciertos casos, impedir la convergencia de algoritmos tradicionales de RL hacia una política óptima.

El incentivo cardinal de este estudio es examinar cómo se desempeñan los algoritmos de RL en problemas caracterizados por su alta dimensionalidad, específicamente en el ámbito de la planificación forestal. Abordar estos problemas con enfoques tradicionales de programación dinámica, métodos heurísticos o técnicas de optimización clásica puede verse afectado por la "maldición de la dimensionalidad"[1]. En este marco, se busca investigar la eficacia de técnicas de RL en la aproximación y solución de este tipo de problemas para diferentes instancias.

1.2. Definición del problema

El problema que se busca abordar en esta tesis corresponde a una variante del problema de planificación de la cosecha en el sector forestal, basado en las formulación presentada "Forestry management under uncertainty"[2].

En este trabajo, el problema de planificación es planteado como un problema de programación estocástica de horizonte multi-etapa. Esto implica tomar decisiones en diferentes momentos a lo largo de un horizonte de planificación, y ajustar esas decisiones a medida que se recibe nueva información. Las decisiones consideradas en este problema incluyen cuándo y dónde realizar la cosecha, siendo sometido a la incertidumbre en los precios de la madera y la demanda de mercado. Estas fuentes de incertidumbre se modelan mediante un árbol de escenarios, que representa diferentes realizaciones posibles de precios a lo largo del tiempo.

Finalmente, el problema está sujeto a diferentes restricciones, tales como las demandas de mercado por madera, los costos operacionales y otras limitaciones inherentes a la estructura

y formulación del problema.

1.3. Preguntas de investigación y objetivos

Tomando en consideración el problema definido en la sección anterior, las preguntas y objetivos que se buscan abordar en esta tesis vienen dados por:

¿Cómo se pueden implementar técnicas de Reinforcement Learning para la resolución de problemas estocásticos de gran dimensión?

- Investigar los desafíos asociados a la alta dimensionalidad de los problemas sujetos a incertidumbre.
- Analizar cómo los algoritmos de RL pueden atacar dichos desafíos.
- Evaluar el rendimiento del uso de RL para resolver problemas estocásticos sujetos a múltiples fuentes de incertidumbre.

¿Cómo se puede formular y resolver el problema de planificación de la cosecha forestal bajo incertidumbre utilizando RL?

- Estudiar las formulaciones existentes para problemas de planificación bajo incertidumbre.
- Identificar cuáles son las variables de decisión y parámetros clave que gobiernan el problema.
- Definir una formulación basada en Reinforcement Learning que logre captar dichas variables y componentes principales del problema.
- Implementar los algoritmos necesarios para resolver las diferentes formulaciones del problema.

¿Cuáles son las implicaciones y beneficios de utilizar RL en la planificación de la cosecha forestal bajo incertidumbre?

- Evaluar el desempeño de los algoritmos de RL con respecto a los enfoques tradicionales de optimización bajo incertidumbre.
- Analizar la eficacia computacional y escalabilidad del uso de RL.
- Evaluar el nivel de robustez de las soluciones al utilizar RL frente a diferentes escenarios o instancias del problema.

Los objetivos primordiales de esta tesis se orientan hacia la exploración de técnicas avanzadas de Reinforcement Learning (RL) en el contexto de problemas estocásticos. Este estudio se concentra específicamente en el ámbito de la planificación de la cosecha forestal, donde se enfrentan incertidumbres asociadas con precios y demandas. Se busca investigar si, y en qué circunstancias, las estrategias basadas en RL pueden ofrecer soluciones que se adapten de manera efectiva a la variabilidad y la incertidumbre inherentes a este sector.

1.4. Descripción general del enfoque

Para los objetivos de investigación, se seguirá el siguiente enfoque:

1.4.1. Marco Teórico

- Estudiar las formulaciones y enfoques existentes para el problema de planificación de cosecha forestal bajo incertidumbre.
- Identificar las brechas y desafíos en la literatura actual relacionada con enfoques basados en RL para problemas de planificación.

1.4.2. Formulación del problema

- En base a la literatura revisada, desarrollar una formulación para la planificación de cosecha forestal bajo incertidumbre que se adapte a las técnicas de RL y los procesos de decisión Markovianos.
- Definir diferentes grados de complejidad en la formulación del problema, considerando la incorporación de diferentes variables de decisión, restricciones y fuentes de incertidumbre.

1.4.3. Diseño e implementación de algortimos de RL

- Implementar diferentes algoritmos que se adecuen al problema dada su naturaleza estocástica.
- Explorar los diferentes algoritmos tales como Q-Learning, Policy Gradient Methods y sus variantes usando Deep Reinforcement Learning.
- Adaptar los algoritmos seleccionados para abordar las características propias del problema.

1.4.4. Evaluación experimental

- Recopilación de datos y generación de escenarios para poner a prueba el rendimiento de los algoritmos implementados.
- Comparar los resultados y rendimiento del enfoque de RL respecto a los métodos existentes.
- Analizar la eficiencia computacional y escalabilidad de los algoritmos utilizados.

1.4.5. Discusión y conclusiones

- Resumir los diferentes hallazgos y resultados encontrados.
- Discutir las implicancias y beneficios de utilizar este enfoque para el problema de planificación forestal bajo incertidumbre.
- Evidenciar los diferentes tipos de limitaciones y las direcciones futuras a investigar.

Siguiendo este enfoque, esta tesis tiene como objetivo explorar como alternativa a las técnicas existentes la aplicación de RL para abordar problemas de planificación forestal, centrado en diferentes fuentes de incertidumbre.

Capítulo 2

Marco Teórico

2.1. Reinforcement Learning (RL)

2.1.1. Definiciones e introducción

El Reinforcement Learning es una disciplina de investigación dentro del campo del aprendizaje automático que se enfoca en cómo los agentes pueden aprender a tomar decisiones óptimas a través de la interacción con su entorno. A diferencia de otros enfoques de aprendizaje automático que se basan en conjuntos de datos de entrenamiento, el RL utiliza las interacciones dentro de dicho ambiente para obtener una retroalimentación que le permite ir aprendiendo. [3]

El objetivo fundamental del RL es desarrollar estrategias de toma de decisiones óptimas que maximicen una medida de recompensa acumulativa. Los agentes de RL buscan descubrir políticas de acción que les permitan obtener la máxima recompensa a largo plazo. Para lograr este objetivo, dichos agentes deben aprender a mapear situaciones o estados específicos a las acciones más apropiadas. [3]

Los elementos fundamentales del RL son los siguientes:

- **Agente**: El agente es la entidad encargada de la toma decisiones en el sistema. Es el responsable de seleccionar acciones basándose en el estado actual y las recompensas recibidas. El objetivo del agente es aprender una política que maximice la recompensa acumulada esperada a lo largo del tiempo.
- **Entorno**: El entorno es el contexto externo con el cual el agente interactúa. Es responsable de proporcionar al agente observaciones, recibir acciones y generar recompensas. Este encapsula la dinámica y las reglas del sistema bajo el cual el agente se debe regir.
- **Acciones**: Son las opciones disponibles para el agente en un momento dado. El agente selecciona acciones basándose en el estado actual y en su política aprendida. Las acciones pueden tener consecuencias a corto plazo y pueden llevar a transiciones a nuevos estados en el entorno.
- **Estados**: Los estados representan las situaciones o configuraciones en las que el agente puede encontrarse durante la interacción con el entorno. El estado encapsula toda la información relevante necesaria para la toma de decisiones. Puede definirse como una descripción completa o parcial del entorno en un paso de tiempo específico.

- **Recompensas**: Corresponde a las señales y feedback proporcionadas por el entorno hacía el agente. Indican la calidad de las acciones del agente y son la guía bajo la cual el agente basa su aprendizaje. El objetivo del agente es maximizar la recompensa acumulada que recibe a lo largo del tiempo.

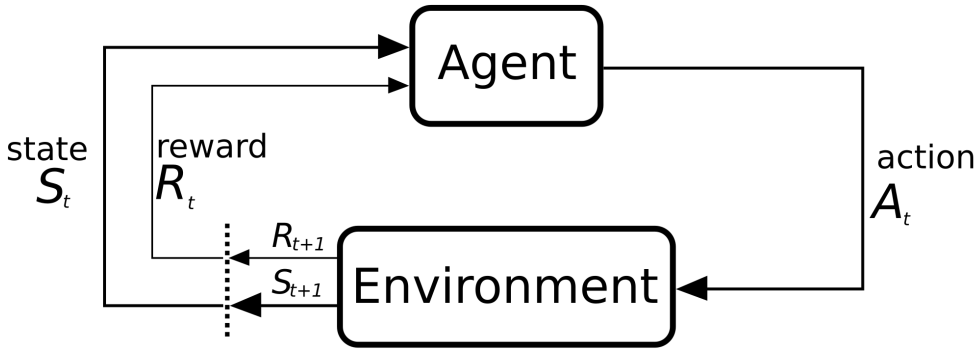


Figura 2.1: Diagrama de RL de un proceso de decisión Markoviano basado en la figura de 'Reinforcement Learning An Introduction' 2da edición de Sutton and Barto.

2.2. Conceptos básicos y técnicas de RL

2.2.1. Funciones de valor

Las funciones de valor representan un concepto fundamental en el ámbito del RL. Definen el retorno esperado (recompensa futura acumulada y descontada) que un agente puede esperar recibir en un estado particular, o después de tomar una acción particular en un estado, bajo una política específica.

La función de valor de un estado s bajo una política π, denotada $v_\pi(s)$, es el retorno esperado comenzando en s y siguiendo π a partir de entonces. La función de valor de acción, denotada $q_\pi(s, a)$, es el retorno esperado después de tomar la acción a en el estado s y a partir de entonces seguir la política π. [3]

Las funciones de valor sirven como medida de la efectividad a largo plazo de los estados y acciones y son fundamentales para que el agente decida qué acción tomar en un estado dado. Tienen una propiedad recursiva derivada de las ecuaciones de Bellman [4], que expresa una relación entre el valor de un estado y los valores de sus estados sucesores.

2.2.2. Política

Una política en RL es un mapeo de estados a probabilidades de tomar cada acción posible. En términos prácticos proporciona una estrategia para el agente, dictaminando la acción que

el agente debe tomar frente a cada estado. La política podría ser determinista, donde la selección de la acción es cierta para cada estado, o estocástica, instancia en la cual las acciones son seleccionadas basándose en una distribución de probabilidad de las acciones para cada estado. [3]

Políticas Deterministas

Una política determinista se puede definir como una función que asigna estados s a acciones a, y se denota como $\pi(s) = a$. Esto se puede representar utilizando una aproximación determinista de funciones, donde la política es una función que produce directamente la acción basada en el estado.

Las políticas deterministas son adecuadas para problemas donde se espera que el agente tome acciones precisas y consistentes en respuesta a un estado dado. Estos problemas generalmente tienen una solución óptima única y determinista.

Políticas Estocásticas

Una política estocástica asocia una distribución de probabilidad sobre las acciones para cada estado. Se puede representar como $\pi(a|s)$, que representa la probabilidad de seleccionar la acción a dado el estado s. Las políticas estocásticas se pueden representar utilizando una función de distribución de probabilidad, como la función softmax, donde la salida es una distribución de probabilidad sobre las acciones. [3]

La función softmax es comúnmente utilizada para definir políticas estocásticas en RL. Dado un estado s y un conjunto de estimaciones de valor de acción $Q(s, a)$ para cada acción a, la distribución de probabilidad sobre las acciones se puede definir de la siguiente manera:

$$\pi(a|s) = \frac{\exp\{Q(s,a)\}/\tau}{\sum_{a'} \exp\{Q(s,a')/\tau\}} \tag{2.1}$$

donde τ es un parámetro que controla el nivel de exploración. Valores más altos de τ generan una distribución más uniforme sobre las acciones, promoviendo la exploración, mientras que valores más bajos hacen que la política se enfoque más en explotar las acciones con valores estimados más altos.

Las políticas estocásticas son beneficiosas en situaciones donde la aleatoriedad puede ser útil para explorar diferentes acciones y estados. Estas políticas son adecuadas para problemas donde la solución óptima puede ser intrínsecamente aleatoria o donde se necesita una exploración amplia del espacio de acciones.

La elección de un tipo de política también depende del equilibrio entre la explotación y la exploración. Las políticas deterministas se centran más en la explotación, ya que tienden a elegir acciones con los valores esperados más altos. Las políticas estocásticas, en cambio, enfatizan la exploración al incorporar aleatoriedad en el proceso de selección de acciones. [3]

2.2.3. Procesos de Decisión de Markov (MDP)

Los Procesos de Decisión de Markov son una herramienta fundamental bajo la cual se sustenta el Reinforcement Learning para modelar problemas de toma de decisiones secuenciales.

En RL, el objetivo es que un agente aprenda a tomar decisiones secuenciales en un entorno sujeto a incertidumbre buscando maximizar su recompensa acumulada a largo plazo. Los MDPs son una representación matemática comúnmente utilizada para modelar este problema. Un MDP se define como una tupla $\langle S, A, P, R, \gamma \rangle$, donde:

- S es el conjunto de estados, que representan las posibles situaciones en las que puede encontrarse el agente.
- A es el conjunto de acciones, que son las opciones disponibles para el agente en cada estado.
- P es la función de transición, que describe las probabilidades de transición entre estados dados los pares estado-acción. Matemáticamente, $P(s'|s, a)$ representa la probabilidad de pasar al estado s' dado que se realiza la acción a en el estado s.
- R es la función de recompensa, que asigna una recompensa numérica al agente en función de los pares estado-acción o estado-siguiente estado. $R(s, a)$ representa la recompensa inmediata obtenida al realizar la acción a en el estado s.
- γ es el factor de descuento, que determina la importancia relativa de las recompensas futuras en relación con las recompensas inmediatas. Un factor de descuento cercano a 1 da más peso a las recompensas a largo plazo, mientras que un valor cercano a 0 enfatiza las recompensas inmediatas.

Los MDPs se rigen bajo de la propiedad de Markov, la cual establece que el estado actual contiene toda la información relevante sobre el pasado para tomar decisiones futuras. Esta propiedad permite que el agente planee su comportamiento de manera óptima utilizando técnicas como la programación dinámica o Reinforcement Learning. [5]

2.2.4. Temporal Difference Learning

Después de haber definido los conceptos básicos de RL, como las funciones de valor, las políticas y los Procesos de Decisión Markovianos (MDP) en las secciones previas, es necesario abordar el Temporal Difference Learning. Este enfoque se basa en los fundamentos previamente establecidos y permite entender cómo se actualizan y aprenden las estimaciones de valor a medida que el agente interactúa con su entorno.

El Temporal Difference Learning (TD) corresponde a una técnica fundamental en el RL, lo que permite que un agente aprenda a través de la experiencia, actualizando sus estimaciones de valor para un estado o acción en base a las recompensas obtenidas. [3]

Concepto de TD-Error: El TD-Error corresponde a una medida que cuantifica la diferencia entre la estimación de valor actual y la estimación de valor anterior para cierto estado o acción. El TD-Error es usado para actualizar las estimaciones de valor y depurar el conocimiento del agente respecto a las recompensas esperadas.

Matemáticamente, el TD-Error se define como la diferencia entre la recompensa actual obtenida y la estimación de valor actual para un estado o acción. Para el caso de los valores de estado, el TD-Error se calcula de la siguiente manera:

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.2}$$

donde δ_t es el TD-Error en el instante de tiempo t, R_{t+1} es la recompensa recibida después de realizar una acción en el estado s_t, $V(s_t)$ es la estimación de valor del estado s_t, s_{t+1} es el estado siguiente después de tomar la acción, y γ es el factor de descuento que determina la importancia relativa de las recompensas futuras.

El TD-Error proporciona una señal de retroalimentación inmediata que indica si las estimaciones de valor son demasiado altas o demasiado bajas. Al actualizar las estimaciones de valor en función del TD-Error, el agente puede aprender a mejorar su comportamiento y ajustar sus acciones en consecuencia.

El TD Learning utiliza el TD-Error en combinación con técnicas de actualización como la actualización de valores Q o la actualización de valores V para mejorar las estimaciones de valor a lo largo del tiempo. Estas actualizaciones se basan en la diferencia entre las estimaciones de valor actuales y las estimaciones de valor anteriores ponderadas por una tasa de aprendizaje.

2.2.5. Q-Learning

Q-Learning [6] corresponde a un algoritmo de RL que es utilizado para encontrar una política óptima. Este se basa en la existencia de una función de valor Q, la cual representa una estimación de cuanto valor tiene cierta acción frente a cierto estado. De esta manera, el Q-Learning es utilizado para aprender dicha función de valor, y luego utilizar su aproximación para tomar decisiones óptimas.

Este corresponde a un método de programación dinámica *off-policy* el cual busca aprender la función de valor de acciones óptimas sobre un ambiente en el cual se deben tomar decisiones de manera secuencial. Para cada par *estado-acción* (s, a), esta función estima el valor esperado de la recompensa total a partir de dicho estado y acción sujeto a una política de carácter optimal. Los beneficios asociados a la utilización de este algoritmo vienen dados por:

1. **Convergencia**: en caso de que se logre explorar de manera adecuada todas las acciones con una frecuencia suficiente, y la tasa de aprendizaje asociada sea consistente, Q-Learning convergerá a la política óptima [3]. Es decir, el algoritmo encontrará la mejor estrategia posible si se le proporcionan las suficientes iteraciones y un grado de exploración adecuado.
2. **Off-policy learning**: Q-Learning tiene la capacidad de aprender en base a las acciones que no son parte de la política que se encuentra siguiente, esto facilita la mejora y ajuste de la política cuando no se esta siguiendo la política óptima.

La actualización de la función Q en Q-Learning se representa matemáticamente de la

siguiente manera:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.3}$$

Donde:

- s_t: Es el estado actual.
- a_t: Es la acción tomada en el estado s_t.
- r_{t+1}: Es la recompensa recibida después de tomar la acción a_t en el estado s_t.
- $\max_a Q(s_{t+1}, a)$: Es el valor máximo de Q sobre todas las acciones posibles en el estado siguiente s_{t+1}.
- α: Es la tasa de aprendizaje, que controla cuánto se modifica la estimación de Q en cada actualización.
- γ: Es el factor de descuento, que determina la importancia de las recompensas futuras versus las inmediatas.

Q-Learning sigue un proceso iterativo para aprender la función de valor Q. Inicialmente, se establece una función Q arbitraria, la cual se va actualizando mediante la ecuación de actualización hasta que la función Q converge al valor óptimo Q^*. El algoritmo Q-Learning [3] se puede describir formalmente de la siguiente manera:

Algoritmo 0: Algoritmo Q-Learning (Sutton & Barto, 2018)

Data: Función de valor Q arbitrariamente inicializada, α (tasa de aprendizaje), γ (factor de descuento)
Result: Función de valor Q aprendida
foreach *episodio* **do**
  Inicializar estado s; **while** s *no es estado terminal* **do**
    Elegir acción a desde s utilizando la política ε-greedy basada en Q;
    Tomar acción a, observar recompensa r, nuevo estado s';
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$;
    $s \leftarrow s'$;
  end
end

Sin embargo, este algoritmo presenta ciertas limitaciones o puntos a considerar al momento de ser utilizado :

1. **Exploración frente a Explotación**: Q-Learning presenta el desafío de equilibrar la exploración (probar nuevas acciones para ver si son mejores) y la explotación (seguir las acciones que se sabe que son buenas). Elegir entre exploración y explotación puede ser muy desafiante en entornos complejos y puede requerir ajustes cuidadosos [7].
2. **Maldición de la dimensionalidad**: Como muchos algoritmos de aprendizaje reforzado, Q-Learning puede sufrir de la "maldición de la dimensionalidad", que se refiere al hecho de que el tiempo y la memoria necesarios para aprender una política óptima aumentan exponencialmente con el número de dimensiones del problema [3]

3. **Dependencia de la inicialización**: Los resultados de Q-Learning pueden ser muy sensibles a la inicialización de la tabla Q. Una mala inicialización puede ralentizar significativamente la convergencia o hacer que el algoritmo converja a una política subóptima [7].

4. **Espacio de acciones continuo**: Q-Learning está basado en la utilización de una tabla Q, la cual presenta una entrada para cada para de estado-acción. Si el espacio de acción es continuo, dicha tabla tendrá una alta dimensionalidad, lo que lo vuelve impráctico [3].

5. **Alta variabilidad de la recompensa**: Para el caso en que la recompensa presenta una alta varianza debido a la incertidumbre asociada a ciertos tipos de problemas, el algoritmo Q-Learning puede tener dificultades para aprender los verdaderos valores de Q, ya que cada actualización de Q se basa en una única muestra de recompensa. Esto puede generar que el proceso de aprendizaje sea inestable y que los valores de Q tengan una alta fluctuación.

2.2.6. Aproximación de funciones

En muchos problemas de RL, el número de estados y/o acciones puede ser enorme o incluso infinito, por lo que no es posible guardar y actualizar una tabla con los valores de todas las posibles combinaciones de estados y acciones. Aquí es donde la Aproximación de Funciones se vuelve crucial.

La Aproximación de Funciones es la técnica utilizada para tratar con este problema de espacio y tiempo. El objetivo es encontrar una función aproximada que pueda representar de manera eficiente el valor de un estado o una acción sin necesidad de almacenar todos los posibles valores [3].

La aproximación de funciones puede usarse tanto para las funciones de valor como para las políticas. En el caso de las funciones de valor, se utilizan para estimar el valor esperado de estar en un estado particular o de tomar una acción en un estado particular. Estas funciones son generalmente representadas por un conjunto de parámetros, que se ajustan para minimizar la diferencia entre los valores estimados y los valores reales.

Por otro lado, la aproximación de políticas directamente aproxima la política óptima. A diferencia de la aproximación de la función de valor, en este caso, los parámetros se ajustan para optimizar directamente alguna medida de rendimiento a largo plazo.

2.2.7. Sobre las ecuaciones de Bellman y Reinforcement Learning

En el dominio del Reinforcement Learning, las Ecuaciones de Bellman se manifiestan como una herramienta analítica fundamental para abordar los desafíos inherentes a la aproximación de funciones. En situaciones donde se emplean modelos de aproximación para encarnar las funciones de valor o las políticas, las Ecuaciones de Bellman proveen un marco teórico robusto para la actualización iterativa y el perfeccionamiento de dichas estimaciones.

Estas ecuaciones son pilares en el estudio de RL y en la teoría de los Procesos de Decisión de Markov (MDP), pues establecen una relación recursiva que permite calcular el valor de un

estado o una acción basándose en los valores de sus respectivos estados o acciones sucesoras.

Bajo la lupa del Reinforcement Learning, el valor de un estado bajo una política específica, simbolizado como $V^{\pi}(s)$, se define como el retorno esperado a partir de dicho estado s, al seguir la política π. Esta definición [3] se expresa mediante la Ecuación de Bellman para las funciones de valor de estado como sigue:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^{\pi}(s')] \tag{2.4}$$

Donde A representa el conjunto de todas las acciones posibles, p es la función de transición de estados, r es la recompensa y γ es el factor de descuento.

Análogamente, la Ecuación de Bellman para las funciones de valor de acción bajo una política determinada, denotada por $Q^{\pi}(s, a)$, se formula de la siguiente manera:

$$Q^{\pi}(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s', a')] \tag{2.5}$$

Se utiliza esta estructura de ecuaciones para actualizar de manera iterativa los valores estimados de los estados y las acciones hasta que estos convergen a su verdadero valor. Bajo ciertas condiciones, esta convergencia está asegurada, permitiendo así que el agente de RL descubra la política óptima.

Las condiciones para la convergencia de los algoritmos de RL que se fundamentan en las Ecuaciones de Bellman son:

- **Estacionariedad de la recompensa y las funciones de transición de estado:** En un proceso de decisión de Markov (MDP), una suposición básica es que las probabilidades de transición y las recompensas son estacionarias, lo que significa que no cambian a lo largo del tiempo. Matemáticamente, esto se puede expresar de la siguiente manera: para todo estado s, acción a, estado sucesor s' y recompensa r, la probabilidad $p(s', r|s, a)$ es independiente del tiempo.
- **Política de exploración continua:** En RL, es fundamental garantizar que todas las acciones sean seleccionadas con una probabilidad no nula para que se evite el problema de converger hacia una política subóptima. Formalmente, esto se puede expresar diciendo que para cualquier política π, para todo estado s y para toda acción a, $\pi(a|s) > 0$. Esta es una forma de asegurar que todas las acciones sean exploradas continuamente.
- **Actualización de la política de acuerdo con un esquema de mejora:** Los algoritmos de RL a menudo utilizan una estrategia de mejora de la política para actualizar sus políticas con base en las estimaciones actuales de las funciones de valor. Una estrategia comúnmente adoptada es la mejora de la política greedy, que actualiza la política para seleccionar la acción que maximiza la función de valor esperada en cada estado. Formalmente, si Q es la función de valor de acción actual, la política mejorada π' se define como $\pi'(s) = \arg\max_a Q(s, a)$ para todo estado s.

Estas condiciones ayudan a garantizar la convergencia de las estimaciones de la función de valor a sus valores verdaderos en los algoritmos de RL que se fundamentan en las Ecuaciones

de Bellman. Esta convergencia es un requisito clave para el descubrimiento de la política óptima en RL.

Por otra parte, para el caso de la optimización de una política, las Ecuaciones de Bellman para las funciones de valor óptimas son las siguientes [3]:

$$V(s) = \max_{a \in A} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{2.6}$$

$$Q(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a' \in A} Q(s', a')] \tag{2.7}$$

Las Ecuaciones de Bellman son fundamentales en una multitud de algoritmos de RL, incluyendo, pero no limitándose, a los métodos basados en diferencias temporales y en programación dinámica [3].

2.3. Policy Gradient Methods

Una vez que hemos establecido la relevancia de las ecuaciones de Bellman en la actualización iterativa de las funciones de valor en el contexto de RL, es útil considerar un enfoque alternativo conocido como *Policy Gradient Methods*. En lugar de centrarse en las funciones de valor, estos métodos buscan optimizar directamente la política con respecto a una medida de rendimiento a largo plazo.

2.3.1. Introducción

Los *Policy Gradient Methods* proporcionan un enfoque para mejorar la política de un agente a través de la optimización por gradientes. A diferencia de los métodos basados en la función de valor, que requieren la estimación de una función de valor y la derivación de una política a partir de esta, los métodos de gradiente de política trabajan directamente con la política, optimizándola respecto a la recompensa esperada acumulada utilizando técnicas de ascenso por gradiente.

2.3.2. Policy Gradients

Estos métodos optan por un enfoque de optimización directa, actualizando la política π del agente por medio del cálculo de gradientes y la posterior aplicación de técnicas de optimización de estos.

El fundamento teórico de estos métodos viene dado por el Teorema del Gradiente de Política [3]. Según este teorema, el gradiente de la función objetivo, es decir, la recompensa esperada acumulativa $J(\theta)$, respecto a los parámetros θ de la política, se puede expresar como una esperanza matemática sobre los gradientes del logaritmo de la política y la función de valor de acción Q^{π_θ}:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)\Big], \tag{2.8}$$

donde:

- $J(\theta)$ representa la recompensa esperada acumulada al seguir la política π_θ.
- $\pi_\theta(s, a)$ es la probabilidad de seleccionar la acción a en el estado s bajo la política π_θ.

- $Q^{\pi_\theta}(s, a)$ es la función de valor de acción, que estima el retorno esperado al seleccionar la acción a en el estado s y seguir luego la política π_θ.
- $\nabla_\theta \log \pi_\theta(s, a)$ es el gradiente del logaritmo de la probabilidad de la política, el cual mide cómo la log-probabilidad de la acción seleccionada varía en función de los parámetros de la política.

Los métodos de Policy Gradient estiman este gradiente y emplean algoritmos de optimización, como el descenso de gradiente estocástico, para actualizar los parámetros de la política θ en la dirección que incrementa la recompensa esperada.

2.3.3. Ventajas de los métodos Policy Gradients

Los métodos de Policy Gradient ofrecen una serie de ventajas únicas en comparación con otras técnicas de RL, que los hacen particularmente adecuados para una amplia gama de problemas:

- **Optimización Directa:** A diferencia de los métodos basados en valor que requieren una etapa de derivación para determinar la política óptima a partir de la función de valor, los métodos de Policy Gradient optimizan la política de manera directa. Esto elimina la necesidad de resolver la ecuación de Bellman, lo que puede ser un desafío en términos computacionales. [8]
- **Capacidad para Manejar Espacios de Acción Continuos y de Alta Dimensión:** Mientras que los métodos basados en valor pueden enfrentar dificultades con los espacios de acción continuos o de alta dimensión debido al problema de la "maldición de la dimensionalidad", los métodos de Policy Gradient son más eficientes en estos escenarios. Esto se debe a que estos métodos representan la política como una función parametrizada, lo que permite manejar espacios de acción de alta dimensión y continuos.
- **Estabilidad y Convergencia:** Los métodos de Policy Gradient presentan buenas propiedades de convergencia y pueden ser más estables que los métodos basados en valor. Esto se debe a que la política se actualiza suavemente a lo largo del gradiente, lo que garantiza que los cambios en la política de un paso a otro son pequeños. Este enfoque puede resultar en un aprendizaje más estable y un menor riesgo de divergencia. [9]
- **Manejo de la Incertidumbre:** Este tipo de métodos son particularmente efectivos en entornos con mucha incertidumbre. Los métodos de Policy Gradient optimizan una esperanza sobre las trayectorias, y por lo tanto, intrínsecamente tienen en cuenta todas las posibles trayectorias que la política puede tomar. Matemáticamente, esto se refleja en el hecho de que la actualización del gradiente de la política implica una suma sobre todas las posibles trayectorias, ponderada por su probabilidad bajo la política actual. En otras palabras, dada una política parametrizada $\pi_\theta(a|s)$, el gradiente de la recompensa esperada acumulada $J(\theta)$ con respecto a θ es:

$$\nabla_\theta J(\theta) = \mathbb{E}\tau \sim \pi_\theta[\nabla_\theta \log \pi_\theta(\tau) R(\tau)]. \tag{2.9}$$

  Aquí, $\nabla_\theta \log \pi_\theta(\tau)$ es el gradiente del logaritmo de la probabilidad de la trayectoria τ bajo la política π_θ, y es una forma de ponderar las trayectorias con respecto a su probabilidad.

La suma sobre todas las posibles trayectorias ponderadas por su probabilidad permite que los métodos de Policy Gradient manejen la incertidumbre inherente en el entorno de RL. Así, los métodos de Policy Gradient pueden manejar la incertidumbre al optimizar directamente la esperanza sobre todas las posibles trayectorias, tomando implícitamente en cuenta la variabilidad y la incertidumbre en la dinámica del entorno. [8]

2.3.4. Desafíos en los métodos de Policy Gradient

Aunque los métodos de Policy Gradient son poderosos y versátiles, también presentan ciertos retos inherentes.

- **Convergencia Lenta:** Los métodos de Policy Gradient pueden ser lentos para converger a la política óptima. Esto se debe a que estos métodos realizan actualizaciones graduales en la dirección del gradiente de la política, y pueden requerir un gran número de episodios de interacción con el entorno para obtener estimaciones precisas del gradiente.
- **Variabilidad de las Estimaciones de Gradiente:** Las estimaciones del gradiente en los métodos de Policy Gradient se basan en muestreos estocásticos del rendimiento de la política, lo que puede llevar a una alta variabilidad en las estimaciones del gradiente. Esta variabilidad puede dificultar la convergencia y provocar inestabilidad en el proceso de aprendizaje.

2.3.5. Actor-Critic Methods

Los métodos Actor-Critic son una clase de algoritmos del RL que combinan elementos tanto de los métodos basados en políticas (como los métodos de Policy Gradient) como de los métodos basados en valor. [10]

En este marco, el actor se refiere a la política parametrizada que está siendo optimizada, mientras que el crítico corresponde a una función de valor que se utiliza para criticar las acciones tomadas por el actor y guiar la dirección de la optimización. De esta forma, los métodos Actor-Crítico tratan de combinar lo mejor de ambos enfoques: la capacidad de los métodos basados en políticas para manejar espacios de acción continuos y de alta dimensión, y la eficiencia de los métodos basados en valor en la evaluación de la política.

En términos formales, el actor y el crítico son funciones parametrizadas, usualmente implementadas como redes neuronales. Dada una política parametrizada $\pi_\theta(a|s)$ y una función de valor $V_\phi(s)$, donde θ y ϕ son los parámetros de la política y la función de valor, respectivamente, la actualización del actor se realiza a lo largo del gradiente de la recompensa esperada acumulada, como en los métodos de Policy Gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi\theta}[\nabla_\theta \log \pi_\theta(a|s) A_\phi(s,a)], \tag{2.10}$$

donde $A_\phi(s,a)$ es la función de ventaja, que mide cuánto mejor es tomar una acción a en un estado s en comparación con el promedio de todas las acciones posibles en ese estado, según la función de valor $V_\phi(s)$. Típicamente, la función de ventaja se calcula como la diferencia entre la recompensa obtenida y la función de valor:

$$A_\phi(s,a) = R(s,a) - V_\phi(s) \tag{2.11}$$

Por otro lado, la actualización del crítico se realiza minimizando el error cuadrático temporal (Temporal Difference error, TD error), una medida del error entre la función de valor actual y la recompensa obtenida más el valor del siguiente estado:

$$\Delta\phi = \nabla_\phi \mathbb{E}_{\pi\theta}[(R(s,a) + \gamma V_\phi(s') - V_\phi(s))^2], \tag{2.12}$$

donde γ es el factor de descuento que determina cuánto valor se le da a las recompensas futuras en comparación con las recompensas inmediatas.

Algoritmo 1: Algoritmo Actor-Critic

Result: Aproximaciones óptimas de la política π_θ y la función de valor V_ϕ
Inicializar los parámetros de la política θ y la función de valor ϕ arbitrariamente;
while *el problema no converge* **do**
  Inicializar el estado s;
  while *s no es terminal* **do**
    Elegir la acción a según la política $\pi_\theta(a|s)$;
    Tomar la acción a, observar la recompensa r y el próximo estado s';
    Calcular el error TD $\delta = r + \gamma V_\phi(s') - V_\phi(s)$;
    Actualizar la función de valor: $\phi \leftarrow \phi + \alpha\delta\nabla_\phi V_\phi(s)$;
    Actualizar la política: $\theta \leftarrow \theta + \beta\delta\nabla_\theta \log \pi_\theta(a|s)$;
    $s \leftarrow s'$;
  end
end

Donde:

- α y β son las tasas de aprendizaje para la función de valor y la política, respectivamente.
- γ es el factor de descuento.
- δ es el error de diferencia temporal (TD error).
- $\nabla_\theta \log \pi_\theta(a|s)$ es el gradiente del logaritmo de la probabilidad de la acción a bajo la política π_θ en el estado s.
- $\nabla_\phi V_\phi(s)$ es el gradiente de la función de valor V_ϕ en el estado s.

El problema converge cuando la diferencia entre los valores de la política y la función de valor en dos iteraciones consecutivas es inferior a un umbral predeterminado.

2.3.6. Proximal Policy Optimization

El algoritmo Prooximal Policy Optimization (PPO, por sus siglas en inglés), propuesta por Schulman et al.(2017) [9], es una técnica de RL que busca resolver los problemas de eficiencia y estabilidad en la actualización de políticas que suelen presentar otros métodos de policy gradient.

La idea principal detrás de PPO es realizar una actualización de la política que no se desvíe demasiado de la política original. Para lograr esto, PPO utiliza una función de objetivo que incluye un término adicional que penaliza las actualizaciones que mueven la nueva política

demasiado lejos de la política antigua.

En PPO, se calcula una relación de probabilidad $r_t(\theta)$ que compara la probabilidad de tomar una acción a bajo la política actual π_θ y la política antigua $\pi_{\theta_{\text{old}}}$ en un estado s. La relación de probabilidad se define como:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{2.13}$$

Además, la ventaja $A(s_t, a_t)$ se estima utilizando el algoritmo Generalized Advantage Estimation (GAE) de Schulman et al. [9], que es un método para calcular estimaciones de ventajas más precisas en Reinforcement Learning.

La función objetivo de PPO se define entonces como:

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{2.14}$$

donde $\text{clip}(x, \min, \max)$ limita el valor de x entre min y max, y ϵ es un hiperparámetro que controla cuánto se permite que la política nueva se desvíe de la antigua.

El algoritmo PPO se puede resumir de la siguiente manera:

Algoritmo 2: Proximal Policy Optimization (PPO)

Result: Política óptima π_θ
Inicializar la política π_θ y la función de valor V_ϕ;
while *no se cumpla la condición de parada* **do**
  Recoger un conjunto de trayectorias siguiendo π_θ;
  Calcular las ventajas $\hat{A}t$ usando GAE;
  Optimizar la función objetivo $L^{CLIP}(\theta)$ mediante gradientes de política con respecto a θ;
  Actualizar la política antigua: $\pi_{\theta_{\text{old}}} = \pi_\theta$;
  Ajustar ϕ para minimizar el error cuadrático medio entre $V_\phi(s)$ y las recompensas observadas;
end

PPO busca mejorar la política de manera iterativa utilizando una versión suavizada de la función objetivo, para evitar cambios demasiado grandes en la política en cada iteración. Sin embargo, esto no garantiza una mejora monótona en cada paso, ya que el algoritmo puede encontrar óptimos locales o situaciones donde las actualizaciones de la política no generen mejoras inmediatas.

Ventajas de PPO en contextos de alta dimensionalidad y estocasticidad

En situaciones de alta dimensionalidad y estocasticidad, los métodos basados en la optimización de la política, como es el caso de PPO, tienen una serie de ventajas en comparación con los enfoques basados en el aprendizaje mediante la función de valor, como Q-learning y DQN:

1. **Robustez en la estimación de la política:** En los métodos de aprendizaje para la funciónd de valor, el rendimiento puede ser muy sensible a los errores en la estimación

del valor de Q. En situaciones de alta dimensionalidad y estocasticidad, puede ser difícil obtener estimaciones precisas de Q. En contraste, PPO estima una política directamente y puede ser más robusto a las incertidumbres inherentes a estos entornos. [8]

2. **Reducción del sesgo de extrapolación:** En los métodos basados en Q-learning, la actualización del valor de Q implica una operación de maximización que puede conducir a un sesgo de extrapolación, especialmente en entornos estocásticos. En cambio, PPO optimiza directamente la política y no sufre de este problema. El sesgo de extrapolación se refiere a la sobreestimación de los valores Q en métodos de Q-learning debido a la operación de maximización en entornos estocásticos. Este sesgo puede conducir a decisiones subóptimas y menos robustas.[11]

3. **Manejo de la exploración y explotación:** PPO introduce una estrategia de exploración suave al limitar la actualización de la política para mantenerla cerca de la política anterior. Esto puede ser especialmente beneficioso en entornos con alta dimensionalidad, donde la cantidad de estados y acciones puede hacer que la exploración sea desafiante.

4. **Estabilidad y eficiencia:** A diferencia de los métodos de política de gradiente como el Actor-Critic que pueden sufrir de oscilaciones y inestabilidades en las actualizaciones de la política, PPO proporciona un equilibrio entre la eficiencia del aprendizaje y la estabilidad de la política. Esto se logra al limitar cuánto puede cambiar la política en cada iteración de aprendizaje, garantizando un cambio suave y estabilidad en la actualización de la política. [9]

5. **Capacidad para manejar problemas continuos y de alta dimensión:** PPO puede trabajar directamente con problemas de acción continua, lo que es útil en entornos de alta dimensión. En contraste, los métodos basados en Q-learning como DQN generalmente requieren una discretización de los espacios de acción, que puede ser impracticable en entornos de alta dimensión. *Las ventajas de PPO en términos de estabilidad y eficiencia se discuten en el propio artículo de PPO "Proximal Policy Optimization Algorithms"de Schulman et al. (2017).*

2.4. Deep Reinforcement Learning (DRL)

El aprendizaje por refuerzo profundo (DRL) se ocupa de optimizar políticas de decisión secuenciales, a menudo expresadas como funciones parametrizadas con pesos θ. En lugar de depender de representaciones manuales de estados (características), DRL emplea redes neuronales profundas para aprender automáticamente estas representaciones de las observaciones sin procesar.

2.4.1. Deep Q-Learning (DQN)

Deep Q-Learning extiende Q-Learning al usar una red neuronal para aproximar la función de valor de acción-estado Q, denotada como $Q(s, a; \theta)$. La actualización de Q se realiza de acuerdo con la ecuación de Bellman:

$$Q(s, a; \theta) \leftarrow Q(s, a; \theta) + \alpha \left[r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta) \right]. \tag{2.15}$$

Aquí, s' es el siguiente estado después de tomar la acción a en el estado s, r es la recompensa recibida, y α es la tasa de aprendizaje. DQN introduce técnicas como la memoria de repetición para romper correlaciones temporales y las redes objetivo para estabilizar las actualizaciones de Q.

Se tiene de esta manera el algoritmo DQN [12]:

Algoritmo 3: Algoritmo de Deep Q-Learning (DQN)

Result: Política óptima π^*
Inicializar red Q con pesos aleatorios θ;
Inicializar red objetivo $\hat{Q}$ con pesos $\theta^- = \theta$;
Inicializar memoria de repetición $\mathcal{D}$;
for *episodio = 1, M* **do**
  Inicializar secuencia s_1 y preprocesar s_1 para obtener x_1;
  for $t = 1, T$ **do**
    Elegir acción a_t con política ϵ-greedy basada en red Q;
    Ejecutar acción a_t en el emulador y observar recompensa r_t y secuencia s_{t+1};
    Preprocesar s_{t+1} para obtener x_{t+1};
    Almacenar transición (x_t, a_t, r_t, x_{t+1}) en $\mathcal{D}$;
    Tomar muestra aleatoria de minibatch de transiciones de $\mathcal{D}$;
    Calcular $\hat{Q}$ para cada transición del minibatch utilizando:

$$y_j = \begin{cases} r_j & \text{si terminal} \\ r_j + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-) & \text{si no terminal} \end{cases} \tag{2.16}$$

    Realizar un paso de gradiente en $(y_j - Q(x_j, a_j; \theta))^2$ con respecto a los parámetros de red θ;
    Cada C pasos, actualizar $\hat{Q} = Q$;
  end
end

2.4.2. Policy Gradients con Deep Learning

Los métodos de Policy Gradient, tal como se menciono en secciones anteriores, representan una clase de algoritmos en el aprendizaje por refuerzo que operan directamente en el espacio de políticas, ajustando los parámetros de la política en la dirección que mejora la recompensa esperada acumulada. Un aspecto crucial que vincula estos métodos con el aprendizaje profundo es la representación de la política.

En el caso de entornos de alta dimensión, que son comunes en las aplicaciones del Reinforcement Learning, la política puede presentar una complejidad significativa. Es posible que los métodos de representación tradicionales no sean suficientes para capturar adecuadamente esta complejidad. Aquí es donde el DL se vuelve indispensable, proporcionando un medio para representar y aprender políticas de alta complejidad utilizando redes neuronales profundas.

De manera más rigurosa, supongamos que se tiene una política $\pi(a|s; \theta)$ está representada

por una red neuronal profunda, donde θ denota el conjunto de parámetros de la red. Nuestro objetivo es encontrar los parámetros óptimos θ^* que maximicen la recompensa esperada acumulada. Esto se puede formular matemáticamente como:

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R_t], \tag{2.17}$$

donde γ es el factor de descuento, R_t es la recompensa obtenida en el tiempo t, y la expectativa $\mathbb{E}\pi\theta$ se calcula sobre las trayectorias generadas por la política π_θ. Los métodos de Policy Gradient actualizan los parámetros θ en la dirección del gradiente de la recompensa esperada, es decir:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \tag{2.18}$$

donde α es la tasa de aprendizaje, y $J(\theta)$ denota la recompensa esperada acumulada. Este gradiente $\nabla_\theta J(\theta)$ puede estimarse a partir de las trayectorias generadas bajo la política actual π_θ.

De este modo, la integración de las redes neuronales profundas en los métodos de gradiente de política permite representar políticas altamente complejas de una manera diferenciable, lo que a su vez posibilita la aplicación de métodos de optimización basados en gradientes para mejorar continuamente la política.

Se tiene el pseudo-código del algoritmo PPO utilizando redes neuronales [9]

Algoritmo 4: Proximal Policy Optimization con Aprendizaje Profundo

Result: Parámetros óptimos de la política θ^*

Inicializar los parámetros de la política θ de una red neuronal profunda, los parámetros de la estimación de ventaja ϕ y los parámetros de la función de valor ψ de una segunda red neuronal profunda;

while *no converja* **do**

  Recolectar un conjunto de trayectorias $\mathcal{D} = \{\tau_i\}$ ejecutando la política actual π_θ de la red neuronal;

  Calcular las recompensas a futuro R_t para cada paso en cada trayectoria;

  Para cada paso en cada trayectoria, calcular las estimaciones de ventaja A_t usando los parámetros ϕ;

  Optimizar los parámetros de la función de valor ψ de la red neuronal mediante regresión sobre el error cuadrático medio con objetivos R_t a lo largo de múltiples épocas;

  for *época = 1, ..., E* **do**

    Para cada minibatch $\mathcal{M} \subset \mathcal{D}$, calcular la proporción $r(\theta)$ y el objetivo de recorte de la función objetivo $\mathcal{L}^{CLIP}(\theta)$;

    Actualizar la política de la red neuronal maximizando $\mathcal{L}^{CLIP}(\theta)$:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}_{\mathcal{M}}[\mathcal{L}^{CLIP}(\theta)], \tag{2.19}$$

    donde α es la tasa de aprendizaje;

  end

end

Capítulo 3

Definición y Formulación del Problema

3.1. MIP: Formulación inicial

El problema que se busca resolver es una adaptación del que se plantea en *Forestry management under uncertainty* [2]. En este problema originalmente se tiene como objetivo principal determinar de manera óptima la cosecha y construcción de caminos en un horizonte de tiempo para diferentes escenarios de precios buscando maximizar los ingresos netos satisfaciendo las diferentes restricciones establecidas.

El modelo que se utilizará en este caso corresponde a una simplificación del problema original, en el cual se considera un área forestal sub-dividida en celdas de cosecha, únicamente se debe encontrar la política óptima de cosecha de las celdas para cierto horizonte temporal, en el cual existen diferentes precios de venta para la madera, y además, cotas de demandas.

3.1.1. Cosecha de Madera - Escenarios de Precio

En la primera formulación del problema, se considera un árbol de escenarios de precios que simula las posibles fluctuaciones en el valor de la madera. Este árbol de escenarios se construye empleando un modelo estocástico basado en el comportamiento histórico del precio de la madera.

Conjuntos

- $\mathcal{T}$ - Horizonte temporal.
- $\mathcal{H}$ - Conjunto de celdas de cosecha.
- $\mathcal{T}^-$ - Conjunto de etapas, excluyendo la última.
- Ω - Conjunto de escenarios.
- $\mathcal{G}$ - Conjunto de grupos de escenarios.
- $\mathcal{G}^t$ - Conjunto de grupos de escenarios en el periodo t, para $t \in \mathcal{T}$.
- Ω^t - Conjunto de escenarios que pertenecen al grupo g, para $g \in \mathcal{G}$.

Parámetros

- a_h^t - Productividad de la celda de cosecha h para el periodo t [m3/Ha].
- A_h - Área de la celda de cosecha h [Ha].
- P_h^t - Costo de cosecha por hectárea para la celda h en el periodo t [dólares/ha].
- Q^t - Costo unitario de producción en el periodo t [dólares/m^3].

Parámetros Estocásticos

- $R^{t,\omega}$ - Precio de venta de la madera en el periodo t bajo el escenario ω. [dólar/m3]
- $Z_{t,\omega}^-$, $Z_{t,\omega}^+$ - Límites inferiores y superiores de la demanda en el periodo t bajo el escenario ω. [m3]

Variables de decisión

- $\delta_h^{t,\omega}$ - Toma el valor de 1 si se cosecha la celda h en el tiempo t bajo el escenario ω; en caso contrario, es 0.
- $z^{t,\omega}$ - Demanda total de madera en el periodo t bajo el escenario ω.

Restricciones

- Restricciones de demanda de madera:

$$Z_{t,\omega}^- \leq z^{t,\omega} \leq Z_{t,\omega}^+, \quad \forall t \in \mathcal{T}, \forall \omega \in \Omega \tag{3.1}$$

- Una celda solo puede ser cosechada una vez en el horizonte temporal:

$$\sum_{t \in \mathcal{T}} \delta_h^{t,\omega} \leq 1, \quad \forall h \in \mathcal{H}, \forall \omega \in \Omega \tag{3.2}$$

- Restricciones de no anticipatividad para las variables binarias:

$$\delta_h^{t,\omega} = \delta_h^{t,\omega'} \quad \forall t \in \mathcal{T}^-, \forall g \in \mathcal{G}^t, \forall \omega, \omega' \in \Omega^g, \omega \neq \omega', \forall h \in \mathcal{H} \tag{3.3}$$

Función Objetivo

$$\max \mathbb{E}\left[\sum_{t \in \mathcal{T}} \sum_{\omega \in \otimes} w^\omega \left(\sum_{h \in \mathcal{H}} R^{t,\omega} z^{t,\omega} - \sum_{h \in \mathcal{H}} P_h^t \delta_h^{t,\omega} A_h - Q^t \sum_{h \in \mathcal{H}} z^{t,\omega} \right)\right] \tag{3.4}$$

Donde w^ω es el peso asignado al escenario ω de tal manera que $\sum_{\omega \in \otimes} w^\omega = 1$. La expresión representa el beneficio neto esperado.

3.2. Markov Decision Process (MDP)

3.2.1. Formulación 1: Cosecha forestal - Escenarios de precios y demandas

Este problema implica la gestión de la cosecha de madera en un bosque que consta de varias parcelas o unidades de cosecha. En cada período de tiempo, se puede decidir cosechar algunas de estas unidades, por simplificación, cada una de estas solo puede ser cosechada una vez durante el horizonte de planificación. Cada parcela tiene un costo asociado de cosecha y produce una cierta cantidad de madera que se puede vender a un precio que varía con el tiempo.

En cada período de tiempo, existen límites en la demanda de madera, lo cual en esta formulación, también se encuentra modelado como un árbol de escenarios en el cual para cada realización de sus nodos se presenta una cota inferior y superior de demanda. Estos límites, así como los precios de la madera, son inciertos y evolucionan a lo largo del tiempo de acuerdo con los escenarios predefinidos.

El objetivo del problema es decidir cuándo cosechar cada parcela para maximizar la ganancia total de la venta de madera, teniendo en cuenta los costos de cosecha y las restricciones presupuestarias y de demanda, además de las distintas realizaciones de los precios definidos en un árbol de escenarios.

Escenarios de precios

Este árbol es una representación que modela la incertidumbre asociada a la evolución de los precios de la madera a lo largo del tiempo. En dicho árbol, cada nodo representa un posible precio de la madera en un momento específico, mientras que cada arista representa una transición potencial del precio de la madera de un momento a otro. La raíz del árbol representa el precio inicial de la madera, y cada nivel del árbol corresponde a un período de tiempo diferente dentro del horizonte de planificación.

Cabe destacar que a partir de cada nodo, existen diferentes caminos equiprobables que conducen a los nodos del siguiente nivel. Estos caminos representan las distintas trayectorias posibles que el precio de la madera podría seguir a lo largo del tiempo, conformando así diversos escenarios para la formulación inicial del problema.

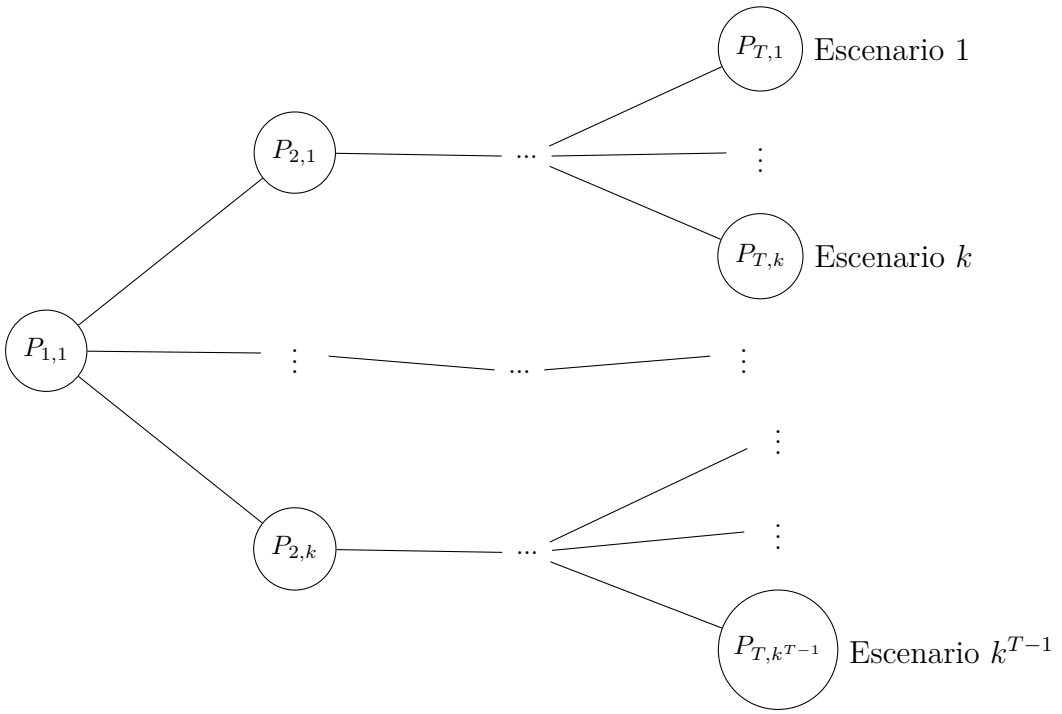


Figura 3.1: Árbol general para k hijos por nodo y un horizonte de tiempo T

En el marco de la formulación como Proceso de Decisión de Markov (MDP) para aplicar algoritmos de RL, el árbol de escenarios de precios se emplea para simular la evolución de los precios de la madera durante el proceso de toma de decisiones. Durante cada paso de tiempo en un episodio, se realiza un muestreo aleatorio de una arista saliente del nodo actual en el árbol de escenarios para determinar el precio de la madera en el siguiente paso de tiempo. Esta fase de muestreo introduce una fuente de aleatoriedad en el entorno, la cual refleja de manera adecuada la incertidumbre inherente a la evolución de los precios de la madera.

Para el caso de este problema, el modelamiento de los precios de la madera mediante un paseo aleatorio geométrico se presenta como una aproximación adecuada para capturar la dinámica de este mercado, debido a su naturaleza estocástica que considera tanto las tendencias deterministas como las fluctuaciones aleatorias.

En este enfoque, se reconoce que los precios de la madera no siguen un comportamiento determinístico simple, sino que están influenciados por diversos factores complejos y a menudo interconectados. El componente determinista del modelo, representado por el término de drift μ, permite considerar la presencia de una tendencia creciente a largo plazo en los precios de la madera. Esto sugiere que, en general, se podría esperar una tendencia ascendente en el valor de la madera a lo largo del tiempo, en línea con factores como el crecimiento económico, el desarrollo de la industria, entre otros.

Por otro lado, las fluctuaciones aleatorias, representadas por el término de volatilidad (σ), capturan las variaciones a corto plazo en los precios de la madera.

Es relevante destacar que el enfoque del paseo aleatorio geométrico no pretende capturar todas las complejidades y particularidades del mercado de la madera, sino que se presenta

como una simplificación que equilibra la representación de las tendencias a largo plazo con las fluctuaciones a corto plazo. Su uso es justificado por su capacidad para proporcionar una descripción razonable del comportamiento general de los precios de la madera en un marco estocástico. Dicho esto, los parámetros que considera la estructuración del árbol de precios corresponden a:

- **Precio inicial** (P_0): Este es el precio de la madera en el momento inicial $t = 0$. En la generación del árbol de escenarios, este es el precio que se asigna al nodo raíz del árbol.
- **Drift** (μ): Este es el término de tendencia del modelo de paseo aleatorio geométrico. Representa el cambio esperado en el precio de la madera por unidad de tiempo, asumiendo que no hay fluctuaciones aleatorias. Esencialmente, μ captura la tendencia sistemática en la evolución del precio a lo largo del tiempo.
- **Volatilidad** (σ): Este es el término de fluctuación aleatoria del modelo de paseo aleatorio geométrico. Representa la desviación estándar de los cambios en el precio de la madera por unidad de tiempo, asumiendo que estos cambios siguen una distribución normal. Es decir, σ encapsula la incertidumbre o el riesgo en la evolución del precio a lo largo del tiempo.
- **Horizonte de tiempo**: (T): Este es el número total de pasos de tiempo en el horizonte de planificación. En la generación del árbol de escenarios, determina la profundidad del árbol, con cada nivel del árbol correspondiente a un paso de tiempo diferente.
- **Hijos por nodo**:(N): Este es el número de nodos hijos que se generan para cada nodo que no es terminal en el árbol de escenarios.

En base a esto, se construye el árbol de escenarios de la siguiente manera:

1. **Inicialización:** Comenzamos con un nodo raíz en $t = 1$ con un precio de inicio P_0. Este nodo se identifica como el nodo 1, es decir, $n = 1$.
2. **Generación del árbol:** Para cada nodo en el nivel t, generamos N nodos hijos en el nivel $t+1$. El precio $P_{i,t+1}$ en cada nodo hijo se genera a partir del precio $P_{n,t}$ en el nodo padre utilizando un modelo de paseo aleatorio geométrico:

   Para cada nodo hijo i, con $i = 1, 2, ..., N$, generamos un camino aleatorio de longitud t con pasos de tamaño $\sqrt{1/t}$ multiplicados por una muestra de una distribución normal estándar. Los pasos se acumulan para formar un camino de precios, y el último precio en el camino es el precio del nodo hijo. Así, el precio del nodo hijo se calcula como:

   $$P_{i,t+1} = P_{n,t} \times \exp\left(\left(\mu - \frac{1}{2}\sigma^2\right) \times \frac{1}{t} \times (t-1) + \sigma \times Z_{i,t+1}\right) \tag{3.5}$$

   donde $Z_{i,t+1}$ es el último paso del camino de precios para el nodo hijo i, μ es el drift (tendencia) y σ es la volatilidad. Los pasos aleatorios $Z_{i,t+1}$ se generan utilizando una semilla aleatoria basada en la identificación del nodo.

   La identificación del nodo hijo se establece como la identificación del nodo padre seguida de ".i".

3. **Probabilidades de transición:** Para cada nodo que no es un nodo terminal, se asignan probabilidades de transición iguales a cada uno de sus nodos hijos. Por lo tanto, la probabilidad de transición a cada nodo hijo es $1/N$.

Este proceso se repite hasta que se alcanza el número especificado de pasos de tiempo T, y todos los nodos en el último nivel se marcan como nodos terminales.

De esta manera, el proceso de generación del árbol de escenarios se basa en la simulación de múltiples caminos posibles para la evolución de los precios de la madera utilizando un modelo de paseo aleatorio geométrico. Este modelo es un modelo estocástico que tiene en cuenta tanto las tendencias deterministas (a través del término de drift μ) como las fluctuaciones aleatorias (a través del término de volatilidad σ). En términos experimentales, la generación del árbol de escenarios es un proceso estocástico controlado utilizando semillas aleatorias basadas en las identificaciones de los nodos, lo que garantiza que el proceso sea reproducible.

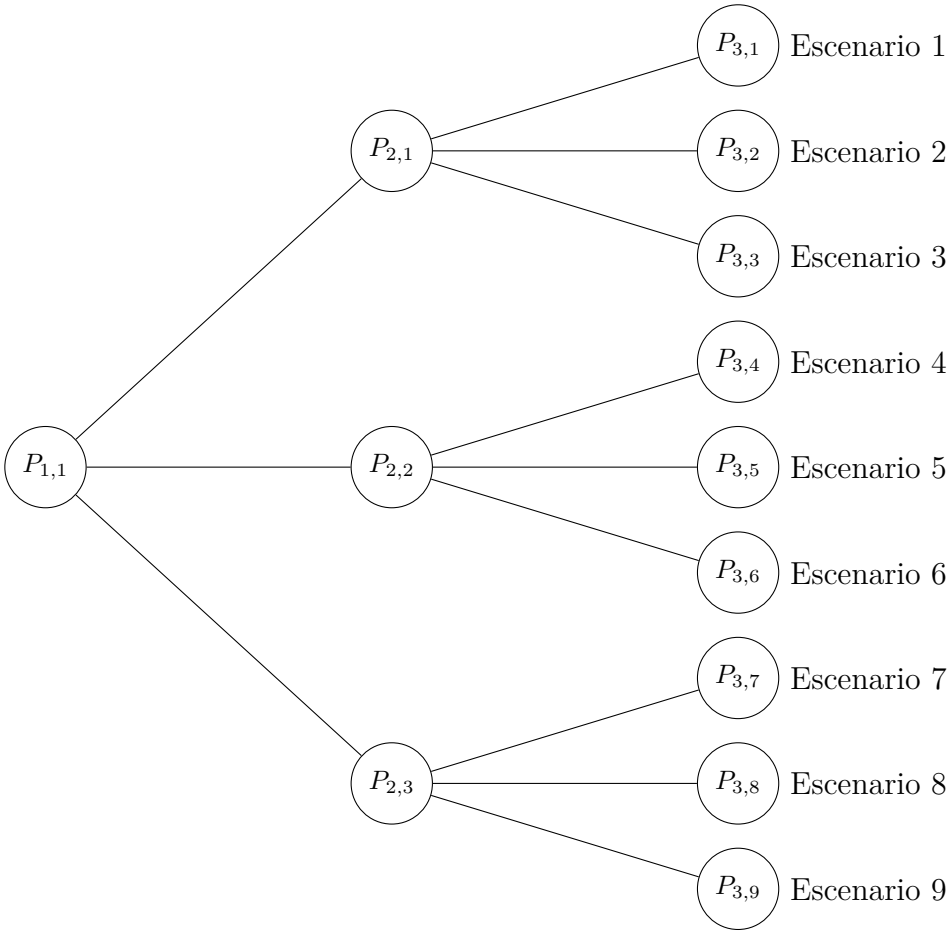


Figura 3.2: Árbol de escenarios de precios de madera. T=3, N=3

Escenarios de demanda

La generación del árbol de escenarios de demanda utiliza un enfoque similar al de precios para modelar la incertidumbre inherente en la evolución de la demanda a lo largo del tiempo. Cada nodo en el árbol representa un intervalo de posibles valores de demanda en un momento específico. La transición de un nodo a otro, representada por una arista en el árbol, corresponde a una evolución potencial de la demanda de un periodo a otro.

La estructura del árbol es tal que la raíz representa el intervalo de demanda inicial, y cada nivel sucesivo del árbol corresponde a un periodo de tiempo subsiguiente dentro del horizonte de planificación.

Para cada nodo, se genera un valor de demanda aleatorio utilizando una distribución normal. Los límites inferiores y superiores del intervalo de demanda para cada nodo se calculan en base a este valor de demanda y una variación aleatoria.

En términos formales, dado un nodo en el tiempo t con un valor de demanda D_t, y una variación aleatoria Z que sigue una distribución normal con media μ y desviación estándar σ, los límites inferiores y superiores del intervalo de demanda, L_t y U_t, se calculan de la siguiente manera:

$$L_t = \max(0, D_t - Z),$$
$$U_t = D_t + Z.$$

En este contexto, la variación aleatoria Z se genera a partir de una distribución normal con una media μ y una desviación estándar predefinida σ. Cada camino desde la raíz hasta un nodo terminal en el árbol representa un escenario posible de demanda a lo largo del tiempo. Por ejemplo, el camino que pasa por los nodos con intervalos de demanda (2911, 5417), (979, 3711), (0, 885) representa un escenario en el que la demanda en el tiempo $t = 1$ está en el intervalo (2911, 5417), la demanda en el tiempo $t = 2$ está en el intervalo (979, 3711), y la demanda en el tiempo $t = 3$ está en el intervalo (0, 885).

3.2.2. Formulación 2: Cosecha forestal: Escenarios de precios y demanda irrestricta

De la misma manera en que se definió la formulación anterior, se considera una variante en la que la demanda no se encuentra restringida.

3.2.3. Definición del problema como MDP

Para poder trasladar el problema al marco del Reinforcement Learning, es necesario modelar el problema original como un proceso de decisión Markoviano. Como se definió en las secciones anteriores, un MDP se define por una tupla (S, A, T, R):

- S es el espacio de estados
- A es el espacio de acciones
- $T(s, a, s')$ es la probabilidad de transición de estado
- $R(s, a, s')$ es la función de recompensa

Dada la naturaleza secuencial del problema, la transformación a un Modelo de Decisión de Markov (MDP) resulta natural. Aunque el espacio de estados incluye elementos históricos como el historial de precios y de demanda, el proceso sigue siendo markoviano. Esto se debe a que, al incluir estos historiales en el estado actual, estamos consolidando toda la información relevante del pasado que podría influir en decisiones futuras dentro del estado presente. En términos formales, la propiedad de Markov sostiene que la decisión óptima en cualquier paso

de tiempo t depende solo del estado actual $s(t)$ y no de la secuencia de estados o acciones previas. Esto se puede expresar como:

$$P(s_{t+1}, r_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(s_{t+1}, r_{t+1}|s_t, a_t) \tag{3.6}$$

Donde P denota la probabilidad, s los estados, a las acciones y r las recompensas. Esta ecuación establece que la probabilidad de transición al siguiente estado s_{t+1} y la recompensa r_{t+1} recibida solo dependen del estado y acción actuales s_t, a_t, y no de la historia que llevó a dicho estado.

Con base en esto, en este problema, las definiciones del espacio de estados, el espacio de acciones, las probabilidades de transición y la función de recompensa se pueden adaptar de la siguiente manera:

Espacio de Estados (S)

Cada estado s para un tiempo determinado t es una tupla compuesta por varios elementos:

- Celdas Cosechadas: Corresponde al vector de decisión para cada paso que da el agente en el ambiente, esta se define como: $H = \{h_1, h_2, \ldots, h_n\}$ donde $h_i \in \{0, 1\}$ indica si la celda i ha sido cosechada.
- Límites de Demanda: D^- y D^+ son los límites inferiores y superiores de la demanda de madera para el periodo de tiempo actual.
- Precio: P es el precio actual de la madera.
- Tiempo: t corresponde al tiempo actual.
- Historial de Precios: $P_{\text{hist}} = \{p_1, p_2, \ldots, p_t\}$ es el historial de precios de la madera para hasta el tiempo t.
- Historial de Demanda: $D_{\text{hist}} = \{(d_1^-, d_1^+), (d_2^-, d_2^+), \ldots, (d_T^-, d_T^+)\}$ es el historial de límites de demanda para cada paso de tiempo.
- Máscara Actual: Para evitar que el agente tome constantemente decisiones inválidas, se genera una mascara que limita las acciones a las celdas disponibles para ser cosechadas, esta máscara se agrega al espacio de estados como información relevante. Esta se define como: $M = \{m_1, m_2, \ldots, m_n\}$ donde $m_i \in \{0, 1\}$ indica si la parcela i puede ser cosechada en el paso de tiempo actual.

Entonces, el espacio de estados es el conjunto de todas las posibles tuplas:

$$S = \begin{pmatrix} H \\ D^- \\ D^+ \\ P \\ T \\ P_{\text{hist}} \\ D_{\text{hist}} \\ M \end{pmatrix} \tag{3.7}$$

Espacio de Acciones (A)

En cada instante de tiempo t, el agente debe seleccionar una acción que corresponde a la decisión de cosechar o no ciertas celdas. Formalmente, una acción $a(t)$ se representa como un vector binario $a(t) = \{a_1(t), a_2(t), ..., a_n(t)\}$, donde n corresponde al número total de celdas.

Cada elemento $a_i(t)$ de este vector de acciones se define como:

$$a_i(t) = \begin{cases} 1 & \text{si se decide cosechar la celda } i \text{ en el tiempo } t, \\ 0 & \text{en caso contrario.} \end{cases} \tag{3.8}$$

Por tanto, la acción $a(t)$ se puede visualizar como un vector de longitud n, donde cada elemento $a_i(t)$ indica la decisión de cosechar (1) o no cosechar (0) la celda i en el tiempo t.

$$A(t) = \begin{pmatrix} a_1(t) \\ a_2(t) \\ \vdots \\ a_n(t) \end{pmatrix} \tag{3.9}$$

De este modo, el espacio de acciones A está compuesto por todos los posibles vectores de acciones que el agente puede seleccionar en cada instante de tiempo.

Función de Transición de Estados (T)

La función de transición de estados $T : S \times A \to S$ describe cómo el sistema evoluciona de un estado a otro dado una acción específica. Esta función se define de la siguiente manera:

- *Cosecha de celdas*: Las celdas que son seleccionadas por la acción son cosechadas, lo que se puede expresar como:

$$h'_i = h_i \vee a_i(t), \quad \forall i \in \{1, \ldots, n\}. \tag{3.10}$$

- *Precio y límites de demanda*: El precio y los límites de demanda se actualizan en función de los árboles de escenario de precios y demanda, respectivamente. En particular, el precio en cada paso de tiempo se muestrea a partir del árbol de precios correspondiente utilizando un enfoque secuencial. Se empieza con el precio inicial en el tiempo $t = 0$, y para cada paso de tiempo subsiguiente, se selecciona un precio hijo del precio actual en el árbol de precios, tomando en consideración el elemento sampleado previamente. Este proceso de muestreo puede expresarse matemáticamente como:

$$P' = f_P(T, P_{\text{hist}}, \text{Árbol de precios}), \tag{3.11}$$

$$D'^{-} = f_D^{-}(T, D_{\text{hist}}, \text{Árbol de demandas}^{-}), \tag{3.12}$$

$$D'^{+} = f_D^{+}(T, D_{\text{hist}}, \text{Árbol de demandas}^{+}), \tag{3.13}$$

donde f_P, f_D^-, f_D^+ son funciones de muestreo que seleccionan un nuevo precio y nuevos límites de demanda a partir del árbol de escenarios correspondiente a cada instante de tiempo. Aquí, Árbol de precios, Árbol de demandas$^-$ y Árbol de demandas$^+$ son los árboles de escenarios de precios y demanda, respectivamente.

- *Tiempo*: El paso de tiempo se incrementa:

$$T' = T + 1. \tag{3.14}$$

- *Historiales de precios y demanda*: El historial de precios y de demanda se actualizan:

$$P'_{\text{hist}} = P_{\text{hist}} \cup \{P'\}, \tag{3.15}$$
$$D'_{\text{hist}} = D_{\text{hist}} \cup \{(D'^{-}, D'^{+})\}. \tag{3.16}$$

- *Máscara actual*: La máscara actual se actualiza en función de las celdas cosechadas:

$$m'_i = \neg h'_i, \quad \forall i \in \{1, \ldots, n\}. \tag{3.17}$$

Así, la función de transición de estados T proporciona una descripción completa de cómo el sistema evoluciona en el tiempo dada una secuencia de acciones.

Función de Recompensa (R)

La función de recompensa, denotada por $R : S \times A \to \mathbb{R}$, se utiliza para evaluar la utilidad que se obtiene al ejecutar una acción específica en un estado determinado. En el problema que nos ocupa, la recompensa se define como la ganancia neta resultante de la cosecha de las celdas especificadas por la acción en el instante de tiempo t. Esta ganancia neta se calcula como la diferencia entre los ingresos generados por la venta de la madera, que se determinan utilizando los precios muestreados del árbol de escenarios de precios, y los costos asociados con la cosecha de la madera, en conjunto con las acciones seleccionadas, las que determinan la cantidad demandada d_i de madera según el rendimiento de cada celda seleccionada.

De manera más formal, la función de recompensa se define como sigue:

$$R(s, a(t)) = \begin{cases} \sum_{i=1}^{n} a_i(t) \cdot (P \cdot d_i - c_i) & \text{si la acción } a(t) \text{ es válida,} \\ -\infty & \text{si la acción } a(t) \text{ es inválida.} \end{cases} \tag{3.18}$$

Una acción $a(t)$ es considerada válida si y solo si cumple con la restricción de demanda definida de la siguiente manera:

- *Restricción de demanda*: El volumen total de madera cosechada, que se calcula como $\sum_{i=1}^{n} a_i(t) \cdot d_i$, debe estar comprendido entre los límites de demanda D^- y D^+. Esta restricción se puede formular como:

$$D^- \leq \sum_{i=1}^{n} a_i(t) \cdot d_i \leq D^+. \tag{3.19}$$

- *Restricción de Cosecha Única por Celda*: Esta restricción garantiza que cada celda solo puede ser cosechada una vez. Formalmente, se describe como:

$$a_i(t) \cdot h_i = 0, \quad \forall i \in \{1, \ldots, n\}. \tag{3.20}$$

  Si $h_i = 1$, indicando que la celda i ya ha sido cosechada, entonces $a_i(t)$ debe ser 0, lo que prohíbe su cosecha adicional. Gracias a la máscara incorporada en el espacio de estados, esta restricción se respeta de manera inherente en las decisiones del agente.

Si una acción no cumple con alguna de estas restricciones, se considera inválida y se le asigna una penalización severa, que se representa mediante el valor $-\infty$ en la Ec. 3.18.

Es importante destacar que en cada instante de tiempo, el precio P se obtiene a través del muestreo del árbol de escenarios de precios correspondiente. Este muestreo permite modelar de manera efectiva la incertidumbre inherente a la evolución de los precios de la madera a lo largo del tiempo. Dado que el proceso de muestreo es independiente del estado y de las acciones del agente, se garantiza que la propiedad de no-anticipatividad [13] se preserva en todo momento. En otras palabras, en ningún momento se utiliza información futura en el proceso de toma de decisiones. Esta característica permite al agente adaptar sus decisiones a las variaciones en los precios de la madera.

La propiedad de no-anticipatividad es un requisito fundamental en la formulación de problemas de decisión secuencial bajo incertidumbre. Esta propiedad asegura que las decisiones presentes no están influenciadas por decisiones futuras. En este caso, la propiedad de no-anticipatividad se manifiesta en la forma en que se seleccionan los precios: a pesar de que el árbol de escenarios de precios contiene información sobre posibles futuros precios, en cada instante de tiempo, la selección del precio depende únicamente del nodo seleccionado en el instante anterior y de las probabilidades asociadas a sus nodos hijos.

Capítulo 4

Metodología

4.1. Solución problema original

4.1.1. Descomposición básica de escenarios

Con el fin de obtener una línea de base respecto a los límites de la recompensa para los escenarios de mayor y menor precio, se realiza inicialmente una resolución individual de cada escenario como un sub-problema. En este enfoque, cada escenario se trata de forma independiente, y se resuelve cada instancia de manera separada. El objetivo es ponderar la función objetivo para cada una de las soluciones de los subproblemas resultantes considerando los pesos equiprobables.

Es importante destacar que esta estrategia relaja las restricciones de no-anticipatividad, lo cual implica que esta solución solo se utiliza como una primera aproximación para conocer los límites de optimalidad independientes en los que vive la función de recompensa según los escenarios.

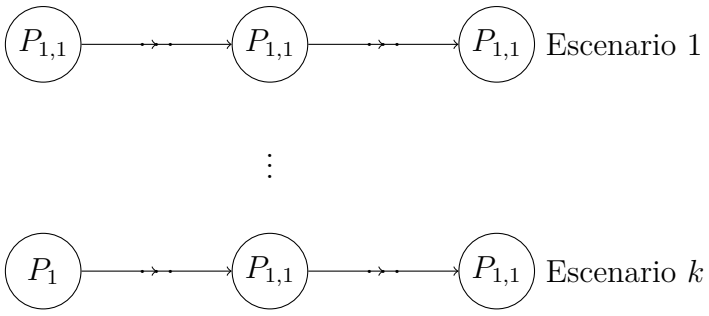


Figura 4.1: Sub-problemas con cada escenario independiente

4.2. Progressive Hedging Cosecha de Madera - Escenarios de Precio

El algoritmo de Progressive Hedging (PH) es una técnica diseñada para resolver problemas estocásticos de programación extensa. PH es un procedimiento de coordinación-descomposición que utiliza penalizaciones para lograr la convergencia hacia una solución estacionaria. [13]

4.2.1. Descripción del método

El algoritmo de Progressive Hedging se basa en resolver una serie de subproblemas (uno para cada escenario) y ajustar las penalizaciones entre iteraciones hasta que la solución converja. La idea es relajar las restricciones de no-anticipatividad, incluyendo penalizaciones que garantizan que las decisiones tomadas en los diferentes escenarios sean consistentes a lo largo de las etapas.

4.2.2. Formulación del subproblema para un escenario ω

Variables de decisión

- $\delta_h^{t,\omega}$ y $z^{t,\omega}$ - Ya definidas anteriormente.

Restricciones

- Restricciones de demanda de madera y una celda cosechada una vez, ya definidas anteriormente.
- Penalización de no-anticipatividad: Introducimos una variable y_h^t que representa la decisión de cosecha anticipada promedio en el período t. La penalización se introduce utilizando una variable dual ρ_h^t y una constante de penalización ρ:

$$\delta_h^{t,\omega} - y_h^t \leq \frac{1}{\rho}\rho_h^t, \quad \forall h \in \mathcal{H}, t \in \mathcal{T}, \omega \in \Omega \tag{4.1}$$

Función Objetivo

$$\max \left[\sum_{t \in \mathcal{T}} w^\omega \left(\sum_{h \in \mathcal{H}} R^{t,\omega} z^{t,\omega} - \sum_{h \in \mathcal{H}} P_h^t \delta_h^{t,\omega} A_h - Q^t \sum_{h \in \mathcal{H}} z^{t,\omega} \right) + \sum_{h \in \mathcal{H}, t \in \mathcal{T}} \rho_h^t \left(\delta_h^{t,\omega} - y_h^t \right) \right] \tag{4.2}$$

4.2.3. Algoritmo

1. Inicialización: Establecer ρ_h^t y y_h^t para todos $h \in \mathcal{H}$ y $t \in \mathcal{T}$.
2. Iteración: Mientras no se cumplan los criterios de convergencia:
   a) Resolver el subproblema para cada escenario $\omega \in \Omega$.
   b) Actualizar las penalizaciones y y_h^t según las soluciones de los subproblemas.
   c) Verificar los criterios de convergencia.
3. Finalización: Una vez que las soluciones de los subproblemas han convergido, la solución general se compone de las decisiones y_h^t promediadas a través de todos los escenarios.

El modelo fue desarrollado utilizando la biblioteca Pyomo, que es una herramienta para la modelización de problemas de optimización. Además, para abordar el aspecto de optimización bajo incertidumbre, se hizo uso de la biblioteca mpi-sppy, que está basada en trabajos recientes para paralelizar el algoritmo de Progressive Hedging [14]. La combinación de estas herramientas facilitó la creación y resolución de nuestro modelo estocástico.

4.3. Algoritmos de RL

Se utilizaron los siguientes algoritmos para el problema:

4.4. DQN

Inicialmente se tiene Deep Q-Network (DQN) que es un algoritmo de Reinforcement Learning [12]. Se construyó con el objetivo de combinar la capacidad de las redes neuronales profundas para manejar espacios de estado de alta dimensión con la eficiencia de los métodos basados en Q-learning para resolver problemas de RL.

Las características clave del DQN incluyen el uso de redes neuronales para aproximar la función de valor Q, el replay experience para abordar las correlaciones temporales y espaciales en los datos de entrenamiento, y las redes objetivo para proporcionar objetivos de entrenamiento estables. [15]

A nivel de la implementación práctica de este algoritmo se generaron un conjunto de clases que interactúan entre si y permiten realizar el entrenamiento, estas corresponden a:

4.4.1. Q-Network

La red neuronal utilizada para aproximar la función de valor-acción, denotada como $Q(s, a)$, que estima el valor esperado de la recompensa acumulativa al realizar una acción a en un estado s y seguir una política óptima después. En el contexto de DRL, la función $Q(s, a)$ que guía al agente para seleccionar las acciones que maximizan la recompensa acumulativa esperada.

La arquitectura de esta red está diseñada para abordar problemas de decisión multidiscreta, aprovechando capas lineales completamente conectadas. La configuración consta de varias capas ocultas, cada una de las cuales utiliza la función de activación ReLU para introducir no linealidad en la red. Esta arquitectura permite que el agente aprenda y represente relaciones complejas entre los estados y los valores de las acciones.

La primera capa oculta recibe el estado de entrada y lo transforma mediante una capa lineal. Esta transformación se puede expresar matemáticamente como:

$$h_1 = \text{ReLU}(W_1 \cdot s + b_1)$$

Donde:

- s es el estado de entrada.
- W_1 y b_1 son los parámetros de la capa (pesos y sesgos, respectivamente).
- $\text{ReLU}(x)$ es la función de activación ReLU, definida como $\max(0, x)$.

La salida de la primera capa oculta (h_1) se pasa a través de capas adicionales, cada una de las cuales también aplica la función de activación ReLU. Esto agrega profundidad a la red y le permite capturar relaciones más complejas.

Finalmente, la última capa lineal (h_4) se conecta a la capa de salida. La capa de salida realiza una transformación lineal adicional, seguida de la función de activación sigmoide para obtener valores de acción. La transformación en la capa de salida se puede expresar como:

$$Q(s, a; \theta) = \sigma(W_4 \cdot h_4 + b_4)$$

Donde:

- $Q(s, a; \theta)$ es la función de valor-acción aproximada.
- θ representa los parámetros de la red, incluyendo los pesos (W_1, W_4) y los sesgos (b_1, b_4).
- $\sigma(x)$ es la función de activación sigmoide, definida como $\frac{1}{1+\exp(-x)}$.

Bajo la premisa de que las decisiones que debe tomar el agente son multidiscretas, se establece una configuración distintiva en la capa de salida. En este escenario, la red neuronal se caracteriza por disponer de una capa de salida que abarca la totalidad de las posibles combinaciones de acciones que el agente puede ejecutar. Cada nodo en esta capa de salida representa una combinación específica de acciones, y la salida correspondiente a cada nodo refleja el valor Q estimado para esa configuración de acciones particular.

De manera más especifica, la capa de salida en la red neuronal está configurada para constar de un número de nodos equivalente a todas las combinaciones viables de acciones que el agente tiene a su disposición, lo cual se traduce en 2^n en esta instancia. Cada nodo de salida corresponde a una combinación única de acciones y la salida resultante aporta una estimación del valor Q asociado con esa particular combinación. Esta estructura confiere al agente la capacidad de evaluar y contrastar diversas combinaciones de acciones en función de sus respectivos valores Q.

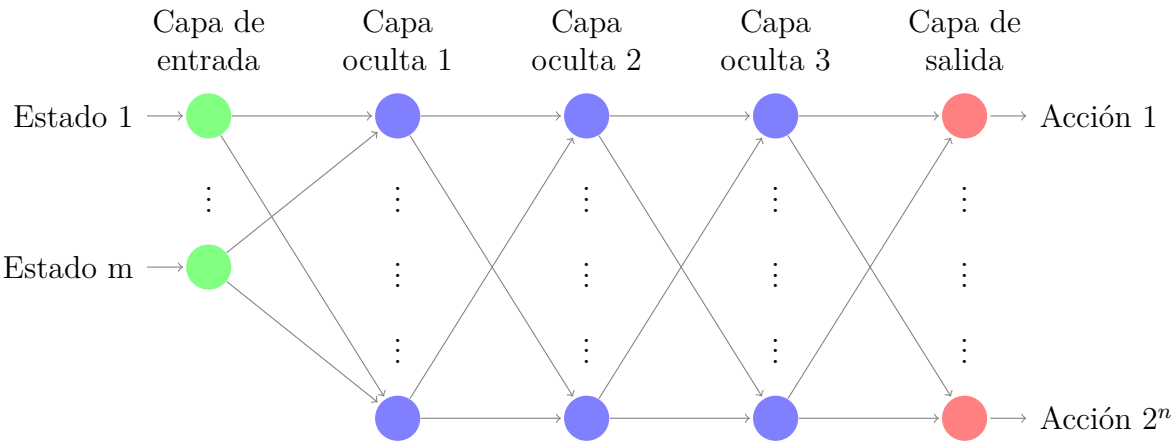


Figura 4.2: Diagrama de la red neuronal Q utilizada en la implementación. La red consta de una capa de entrada, tres capas ocultas con activación ReLU, y una capa de salida con activación sigmoide. m corresponde al tamaño de las observaciones, y n a la dimensión del vector de acciones en formato binario.

4.4.2. Replay Buffer

El Replay Buffer se utiliza para almacenar las experiencias pasadas del agente y proporcionar una muestra aleatoria de estas experiencias para el entrenamiento, una experiencia es una tupla que consta de un estado, una acción, una recompensa, un estado siguiente y un indicador de terminación, denotada como $(s, a, r, s', \textbf{\textit{done}})$.

La capacidad del Replay Buffer, denotada como C, es un hiperparámetro que se elige según los recursos de memoria disponibles y los requisitos del problema. El Replay Buffer almacena las experiencias del agente en un búfer circular, que es un tipo especial de estructura de datos que elimina automáticamente los elementos más antiguos cuando se alcanza la capacidad.

El Replay Buffer se utiliza durante el entrenamiento para muestrear un lote de experiencias para la actualización de los parámetros de la red. Dado un tamaño de lote B, se seleccionan aleatoriamente B experiencias del Replay Buffer. Este proceso de muestreo aleatorio es fundamental para evitar cualquier correlación entre experiencias consecutivas, lo cual es importante para mejorar la estabilidad del entrenamiento y evitar la oscilación o la divergencia de los parámetros de la red.

4.4.3. Prioritized Experience Replay (PER)

El *Prioritized Experience Replay* (PER) es una mejora del concepto tradicional de *Replay Buffer* en el aprendizaje por refuerzo. A diferencia del muestreo uniforme de experiencias, el PER asigna una prioridad a cada experiencia, basada en su relevancia, permitiendo un muestreo sesgado que favorece las experiencias más significativas para el entrenamiento del agente.

4.4.3.1. Concepto de Prioridad

En el PER, cada experiencia $(s, a, r, s', done)$ es asignada con una prioridad. Esta prioridad suele ser una función del error de predicción o la magnitud del error de Diferencia Temporal (TD error), reflejando la cantidad de aprendizaje potencial que la experiencia puede aportar. Las experiencias con altas prioridades tienen una mayor probabilidad de ser seleccionadas para el entrenamiento.

4.4.3.2. Estructura de Almacenamiento

Para facilitar un muestreo eficiente basado en las prioridades, el PER utiliza una estructura de almacenamiento especializada como un *SumTree*. Un *SumTree* es un árbol binario donde cada nodo almacena la suma de las prioridades de sus nodos hijos, permitiendo una búsqueda rápida y eficiente de experiencias basadas en sus prioridades acumulativas.

4.4.3.3. Muestreo de Experiencias

Durante el entrenamiento, se selecciona un conjunto de experiencias basándose en sus prioridades. El proceso de muestreo está diseñado de tal manera que las experiencias con mayores prioridades son más propensas a ser elegidas, aunque todas las experiencias tienen alguna probabilidad de ser seleccionadas. Esto asegura que el agente aprenda no solo de las experiencias más recientes o importantes, sino de una variedad representativa de su historial de aprendizaje.

4.4.3.4. Actualización de Prioridades

Las prioridades de las experiencias en el PER se actualizan después de cada paso de entrenamiento, en función de los cambios en el error de predicción. Esto asegura que las prioridades reflejen continuamente el valor de aprendizaje actual de cada experiencia.

4.4.3.5. Beneficios del PER

El uso del PER en el aprendizaje por refuerzo conduce a una convergencia más rápida y eficiente, ya que el agente se enfoca en aprender de las experiencias más informativas. Además, ayuda a mitigar problemas asociados con la correlación de experiencias y la no estacionariedad de los datos, que son comunes en los métodos de muestreo uniforme.

4.4.3.6. Consideraciones de Implementación

La implementación del PER implica consideraciones adicionales en comparación con un *Replay Buffer* tradicional. Estas incluyen la gestión de la estructura de datos para el almacenamiento y la actualización eficiente de las prioridades, así como la adaptación del proceso de muestreo para incorporar las prioridades. Estas características hacen del PER una herramienta poderosa, pero más compleja en el contexto del aprendizaje por refuerzo.

4.4.4. Estrategia Epsilon-Greedy

Epsilon-Greedy es una estrategia de selección de acciones que equilibra la exploración y la explotación durante el proceso de aprendizaje del agente. Esta estrategia se basa en un parámetro de exploración, denotado como ϵ, que determina la probabilidad de que el agente tome una acción aleatoria (exploración) en lugar de seleccionar la acción que maximiza el valor Q estimado (explotación).

El valor de ϵ se inicializa a un valor alto, denotado como $\epsilon_{\text{inicial}}$, para fomentar la exploración al comienzo del aprendizaje. Sin embargo, ϵ disminuye exponencialmente a lo largo del tiempo hasta un valor mínimo, denotado como ϵ_{final}, de acuerdo con una tasa de decaimiento, denotada como $\epsilon_{\text{decaimiento}}$.

Selección de Acciones con Estrategia Epsilon Greedy y Máscara

Para el entrenamiento de la red se considera la seleccion de acciones de la siguiente manera. Supongamos el tamaño del espacio de acciones es N y estamos simulando T pasos de tiempo. A continuación, describiremos cómo funciona la estrategia Epsilon Greedy modificada en este contexto.

- **Inicialización**
  - Tamaño del espacio de acciones: N
  - Máscara de episodio: [1, 1, ..., 1] (para las 2^N posibles acciones correspondientes a [0,0,...,0], [0,0,...,1], ..., [1,1,...,1])
- **Paso de Tiempo** t
  - **Selección de Acción:** La estrategia Epsilon Greedy elige una acción basada en las estimaciones Q actuales. Si el valor aleatorio generado es menor que la tasa de exploración ($rate >$ valor aleatorio), se realiza exploración y se elige una acción aleatoria válida. De lo contrario, se explota la acción con el mayor valor Q estimado.

- **Actualización de la Máscara de Episodio:** Después de seleccionar una acción en el paso de tiempo t, la máscara de episodio se actualiza para reflejar que la acción ha sido tomada. Las acciones que tienen conflictos con la acción seleccionada se marcan como no válidas en la máscara. De esta manera de reduce el espacio de acciones a aquellas que tengan validez para el caso de cosechar o no una celda.
- **Resumen de Acciones Tomadas:** Para cada paso de tiempo t, se registra la acción seleccionada. La máscara de episodio asegura que las acciones conflictivas con las acciones previamente tomadas queden desactivadas.

- **Final del Episodio:** Al final del episodio, la máscara de episodio no se utiliza nuevamente, ya que el agente comienza un nuevo episodio con todas las acciones válidas activadas.

Este enfoque garantiza que el agente cumpla con las restricciones de selección de acciones y respete las acciones tomadas previamente. Cada acción se elige de manera equilibrada entre exploración y explotación, y la máscara de episodio garantiza que las acciones incompatibles no sean seleccionadas.

1. **Inicialización de la Q-Network y Target Network**: Se inicializan dos redes neuronales con el tamaño del espacio de estados y de acciones. Se carga el estado inicial de la Q-Network en la Target Network.

2. **Inicialización del Replay Buffer y la estrategia epsilon-greedy**: Se inicializa el Replay Buffer con la capacidad definida por el hiperparámetro **Capacidad del buffer de repetición**. La estrategia epsilon-greedy se inicializa considerando los hiperparámetros **Epsilon inicial**, **Epsilon final**, y **Tasa de decaimiento de epsilon**.

3. **Inicialización del optimizador**: Se utiliza el optimizador Adam con la tasa de aprendizaje definida por el hiperparámetro **learning rate**.

4. **Entrenamiento**: El bucle principal de entrenamiento consiste en resetear el estado del entorno y luego repetir los siguientes pasos hasta que se alcance una condición de parada:

   - Observar el estado actual.
   - Seleccionar y ejecutar una acción.
   - Recibir la recompensa y el estado siguiente.
   - Almacenar la experiencia en el Replay Buffer.
   - Actualizar la Q-Network si la cantidad de experiencias en el Replay Buffer es mayor que el **Tamaño del lote de experiencia**.
   - Actualizar la Target Network cada ciertos pasos especificados por el hiperparámetro de **Frecuencia de actualización de la red objetivo**.
   - **Procesamiento de Acciones y Optimización**:
     - Convertir las acciones multidiscretas a índices.
     - Calcular los valores Q actuales utilizando los índices de las acciones.
     - Calcular los valores Q siguientes utilizando la Target Network.
     - Encontrar los valores Q máximos para los estados siguientes.

- Calcular los valores Q objetivo para las acciones tomadas.
- Calcular la pérdida utilizando el error cuadrático medio entre los valores Q actuales y los valores Q objetivo.
- Optimizar el modelo utilizando el optimizador definido, en este caso, Adam.

4.5. Dueling DQN

4.5.1. Arquitectura de la Dueling Deep Q-Network

La Dueling Deep Q-Network (Dueling DQN) es una extensión de la arquitectura de la red Q-Network que descompone explícitamente la función de valor-acción $Q(s, a)$ en dos componentes, la función de valor $V(s)$ y la función de ventaja $A(s, a)$. La arquitectura se presenta de la siguiente manera [16]:

1. **Value Stream**: El flujo de valor consta de tres capas densamente conectadas, con el objetivo de aproximar la función de valor $V(s)$, que cuantifica el valor de estar en un estado s. La formulación matemática para el flujo de valor es:

$$\begin{aligned} v_1 &= \text{ReLU}(\text{BN1}(W_1^v \cdot s + b_1^v)) \\ v_2 &= \text{ReLU}(\text{BN2}(W_2^v \cdot v_1 + b_2^v)) \\ V(s) &= W_3^v \cdot v_2 + b_3^v \end{aligned} \tag{4.3}$$

2. **Advantage Stream**: El flujo de ventaja, paralelo al flujo de valor, consta de tres capas densamente conectadas. Su objetivo es calcular la función de ventaja $A(s, a)$, que representa el valor adicional de tomar una acción específica a en el estado s:

$$\begin{aligned} a_1 &= \text{ReLU}(\text{BN1}(W_1^a \cdot s + b_1^a)) \\ a_2 &= \text{ReLU}(\text{BN2}(W_2^a \cdot a_1 + b_2^a)) \\ A(s, a) &= W_3^a \cdot a_2 + b_3^a \end{aligned} \tag{4.4}$$

3. **Combinación de Value y Advantage Streams**: La arquitectura combina el flujo de valor y el flujo de ventaja para calcular la función de valor-acción $Q(s, a)$ utilizando la siguiente expresión:

$$Q(s, a) = V(s) + (A(s, a) - \text{mean}(A(s, a))) \tag{4.5}$$

Esta combinación permite una representación más flexible de la función de valor-acción, desacoplando la valoración del estado de la evaluación de las acciones individuales.

La arquitectura de Dueling DQN utiliza capas lineales completamente conectadas, y la función de activación ReLU se aplica después de cada capa lineal, excepto en las capas de salida del flujo de valor y ventaja. Además, la normalización por lotes se emplea después de las capas lineales y antes de la activación ReLU, lo que a priori se debe reflejar en una mejora sobre la estabilidad del entrenamiento.

La arquitectura está diseñada para manejar un espacio de acción binario multidimensional, representando un total de 2^n combinaciones posibles de acciones, donde n es el tamaño

de la acción. Esta estructura permite una representación y comparación detallada de las combinaciones de acciones.

El empleo de una arquitectura de Dueling DQN permite que el modelo separe explícitamente la evaluación de los estados y las acciones individuales. Esto conduce a una mayor sensibilidad en la estimación de los valores de acción.

Acerca de la Normalización por Lotes (BN)

La Normalización por Lotes (BN) es una técnica utilizada en las capas intermedias de la arquitectura de Dueling DQN. Su función es normalizar las activaciones dentro de un lote, estabilizando así la distribución de las entradas en las capas posteriores. Esto mejora la convergencia y estabilidad del entrenamiento. La incorporación de BN en la arquitectura es una práctica común en el entrenamiento de redes neuronales profundas.

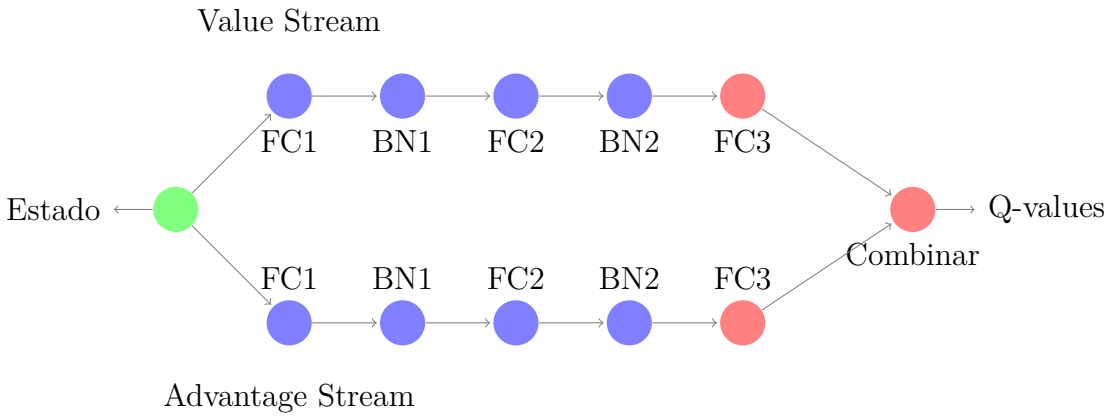


Figura 4.3: Diagrama de la red neuronal Dueling DQN. La arquitectura consta de una capa de entrada que se divide en dos flujos separados: el Value Stream y el Advantage Stream. Cada flujo pasa por capas ocultas y culmina en una capa de salida. Los Q-valores se obtienen combinando las salidas de estos dos flujos.

4.6. Double Dueling DQN

La arquitectura de Double Dueling DQN combina las ventajas del Dueling DQN, previamente detallado, con la técnica del Double Q-learning. Mientras que la estructura de Dueling DQN se mantiene inalterada, el proceso de actualización de los Q-valores es modificado para incorporar el algoritmo Double DQN.[17]

4.6.1. Actualización de Q-valores en Double Dueling DQN

En un DQN estándar, la acción seleccionada durante la actualización de los Q-valores se obtiene utilizando la red Q actual, y el valor de esa acción también es evaluado con la misma red. Sin embargo, esta estrategia a menudo conduce a una sobreestimación de los Q-valores.

El algoritmo Double DQN aborda este problema seleccionando la acción con la red Q actual, pero evaluando esta acción con una red Q objetivo. Esta decoupling de selección y evaluación reduce la sobreestimación y resulta en una mayor estabilidad durante el entrenamiento.

Matemáticamente, la actualización de los Q-valores en Double Dueling DQN se define como:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q^{\text{objetivo}}(s', \text{argmax}_{a'} Q(s',a')) - Q(s,a) \right] \tag{4.6}$$

donde Q^{objetivo} es la red Q objetivo y Q es la red Q actual. El término $\text{argmax}_{a'} Q(s',a')$ determina la acción que maximiza los Q-valores en el siguiente estado s' utilizando la red Q actual.

4.7. PPO

4.7.1. Características Clave del Proximal Policy Optimization (PPO) en Reinforcement Learning

El algoritmo Proximal Policy Optimization (PPO) presenta atributos distintivos que desempeñan un papel fundamental en su aplicación en el contexto del Reinforcement Learning. Estas características esenciales son:

1. Enfoque de Modificación Gradual con Clipping en PPO: PPO se destaca por su estrategia de ajustar la política gradual y controladamente durante la optimización. Una característica esencial de PPO es el uso del clipping en las actualizaciones de la política. Mediante este enfoque, se limita el rango de cambios en la política, asegurando que las actualizaciones permanezcan cercanas a la política previa. Esta técnica evita cambios abruptos que podrían desestabilizar el proceso de aprendizaje. En lugar de introducir ajustes drásticos, PPO aplica ajustes moderados y bien controlados para mejorar progresivamente el rendimiento del agente, garantizando al mismo tiempo la estabilidad del proceso.

2. Exploración Gestionada con Regularización de Entropía: PPO aborda la esencia de la exploración en Reinforcement Learning al incorporar un parámetro de regularización de entropía en su función objetivo. Esto equilibra de manera efectiva la necesidad de explorar nuevas acciones con la importancia de explotar decisiones óptimas. La entropía ajustada guía la exploración sistemática del espacio de acciones, evitando movimientos extremadamente aleatorios que puedan ser perjudiciales para el aprendizaje.

3. Replay Buffer para Almacenamiento de Transiciones: al igual que en DQN, se adopta el uso de un Replay Buffer, una estructura de almacenamiento que captura y retiene las transiciones experimentadas por el agente durante la interacción con el entorno. Esta estrategia optimiza la utilización de los datos recolectados al permitir que las muestras de experiencia sean reutilizadas en múltiples actualizaciones, mejorando la eficiencia del aprendizaje.

4. Aproximación de Funciones de Actor y Valor: PPO emplea redes neuronales para aproximar tanto la política de acciones como la función de valor correspondiente. Esta característica permite mapear relaciones complejas entre estados y acciones. La aproximación de la función de valor también posibilita una estimación de recompensas futuras, fundamental para la optimización a largo plazo de las políticas.

4.7.2. Arquitectura de las Redes en la Implementación PPO

La implementación utiliza dos redes neuronales, estas corresponden a la Actor Network y la Value Network, que desempeñan roles fundamentales en la toma de decisiones del agente y la estimación de las recompensas asociadas.

Actor Network

Esta red mapea las observaciones del entorno a una distribución de probabilidades sobre las posibles acciones. En términos más técnicos, esta red neuronal utiliza capas densas para transformar las observaciones en una representación interpretable por el agente como una distribución de probabilidades. La elección de activación es la función sigmoide, que ajusta los valores de salida entre 0 y 1, lo que es adecuado para modelar probabilidades.

- Varias capas ocultas: La red del Actor utiliza varias capas ocultas, cada una de las cuales introduce no linealidad y capacidad de representación.
- Función de activación ReLU: La función de activación ReLU se aplica en cada capa oculta para introducir no linealidad y aprender relaciones complejas.
- Capa de salida sigmoide: La capa de salida utiliza la función de activación sigmoide, lo que garantiza que las salidas estén en el rango [0, 1], coherente con la naturaleza probabilística de la política.

Value Network

Esta red se dedica a estimar la función de valor asociada a un estado particular. Esta función de valor captura cuán favorable es un estado en términos de las recompensas futuras esperadas. La arquitectura de la red del Valor se basa en capas densas que procesan las observaciones y calculan la estimación del valor de un estado.

- Capas ocultas similares: Al igual que la red del Actor, la red del Valor emplea capas ocultas con función de activación ReLU.
- Capa de salida lineal: La capa de salida realiza una transformación lineal adicional para calcular la estimación del valor, un número real que representa la utilidad esperada del estado.

Ambas redes contribuyen a la optimización de la política en PPO. La red del Actor influye en la toma de decisiones a través de la política estimada, mientras que la red del Valor proporciona información para el cálculo de ventajas y recompensas futuras. Esta combinación de redes permite que el agente ajuste su comportamiento a medida que interactúa con el entorno.

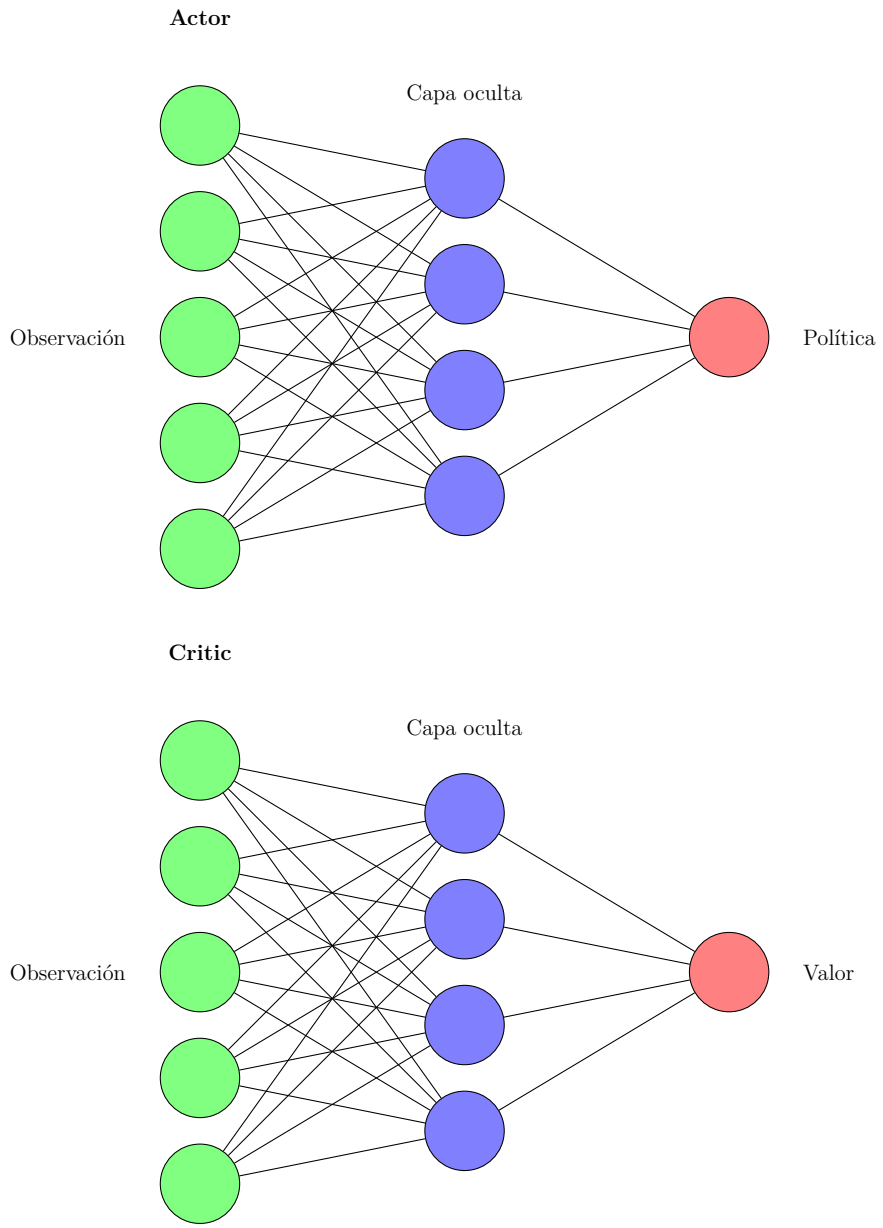


Figura 4.4: Redes Actor y Crítico

4.7.3. Detalle de Parámetros en la Implementación de PPO

Para la implementación del algoritmo Proximal Policy Optimization (PPO) se utilizó la librería tf-agents [18], A continuación, se detallan los hiperparámetros involucrados:

- **Optimizador**: Este parámetro especifica el algoritmo de optimización matemática empleado para la actualización de los parámetros en las redes Actor y Value Network. Comúnmente, se emplean optimizadores estocásticos como Adam o SGD.
- **Actor Network**: Refiere a la estructura de la red neuronal que se encarga de aproximar la política del agente. La arquitectura de esta red debe ser coherente con las dimensiones y características del espacio de acciones.
- **Value Network**: Hace referencia a la arquitectura de la red neuronal destinada a apro-

ximar la función de valor. Esta red se encarga de evaluar las expectativas de recompensas futuras para estados específicos.

- **Regularización**: Este hiperparámetro, comúnmente denotado como β, actúa como un coeficiente de regularización de entropía. Este coeficiente pondera la contribución de la entropía en la función objetivo, permitiendo mantener un equilibrio entre la exploración y explotación.
- **Ratio Clipping**: Este parámetro, denotado como ϵ, limita la razón de importancia en el cálculo del gradiente de la política. Su dominio típico se encuentra en el intervalo $[0, 1]$.
- **Normalización de Observaciones**: Variable booleana que indica si las observaciones del entorno deberán ser normalizadas para tener media cero y desviación estándar uno.
- **Normalización de Recompensas**: Similar al anterior, es una variable booleana que establece si las recompensas capturadas deben ser normalizadas.
- **Algoritmo de Ventaja Generalizada (GAE)**: Es un indicador booleano que señala si se utilizará el algoritmo de ventaja generalizada para calcular la función de ventaja durante la actualización de la política.
- **Número de Épocas**: Denotado como N, este valor entero positivo especifica la cantidad de iteraciones que se realizarán a través del dataset recolectado para cada actualización de la política.
- **Uso de λ-TD**: Indicador booleano que establece si se implementarán los retornos λ-TD para el cálculo de la función de ventaja y las actualizaciones de la red de valor.

El algoritmo PPO es una mejora de los algoritmos de política basados en Trust Region Policy Optimization (TRPO) [19]. PPO es más sencillo de implementar y computacionalmente más eficiente que TRPO, mientras mantiene un buen rendimiento en una variedad de tareas RL.

4.8. Datos utilizados: Esquema del bosque

Para utilizar los diferentes algoritmos de RL, previamente se deben tener en cuenta cuáles son las configuraciones iniciales que se utilizaran para definir el bosque bajo el cual se quiere resolver el problema.

De esta manera, se toma en consideración la siguiente configuración forestal para realizar los diferentes experimentos. Se utilizan los siguientes parámetros para modelar el bosque

- **Número de parcelas,** n: El número de parcelas forestales en consideración. Representa la cantidad total de unidades discretas de área que componen el bosque y que se consideran para la cosecha de madera.
- **Área por parcela,** A_s: El tamaño de cada parcela, asumiendo que todas las parcelas tienen el mismo tamaño. Esta es la superficie unitaria de cada parcela forestal en hectáreas.

- **Productividad por hectárea,** r_s: La cantidad de madera (en metros cúbicos) que se espera obtener de una hectárea de parcela en un período de tiempo. Este valor puede variar de una parcela a otra y está sujeto a incertidumbre.
- **Tasa de crecimiento,** g: La tasa a la que se espera que crezca la producción de madera en cada parcela de un período de tiempo a otro. Este valor es constante en todos los períodos de tiempo y parcelas.
- **Costo de cosecha por volumen de madera,** $c_{\mathbf{vol}}$: El costo de cosechar un metro cúbico de madera. Este valor es constante para todas las parcelas y períodos de tiempo.
- **Costo de cosecha por área,** $c_{\mathbf{área}}$: El costo de cosechar madera de una hectárea de parcela, independientemente del volumen de madera obtenido. Este valor es constante para todas las parcelas y períodos de tiempo.
- **Costo fijo,** $c_{\mathbf{fijo}}$: Un costo que se incurre en cada parcela y período de tiempo, independientemente de la cantidad de madera cosechada. Este valor es constante para todas las parcelas y períodos de tiempo.
- **Tasa de descuento,** d: La tasa a la que se descuentan los costos futuros para llevarlos a términos del valor presente. Este valor es constante en todos los períodos de tiempo y parcelas.
- **Demanda mínima relativa,** $\mathbf{dem_{min}}$: La demanda mínima de madera como proporción de la producción total en el primer período de tiempo. Este valor es constante en todos los períodos de tiempo.
- **Demanda máxima relativa,** $\mathbf{dem_{max}}$: La demanda máxima de madera como proporción de la producción total en el primer período de tiempo. Este valor es constante en todos los períodos de tiempo.

En base a esto, creamos los siguientes conjuntos que permiten instanciar el problema

4.8.1. Área Total del Bosque

La superficie total del bosque, A_{total}, se puede calcular como el producto del número total de parcelas, n, y el área de cada parcela, A_s, que se supone constante para todas las parcelas. Esto se puede representar como:

$$A_{total} = n \times A \tag{4.7}$$

4.8.2. Área por celda en cada periodo

El área de cada parcela en cada período de tiempo, $A_{s,t}$, se mantiene constante a lo largo del tiempo si no se consideran actividades de deforestación o plantación. Por lo tanto, tenemos:

$$A_{s,t} = A_s \quad \forall s \in S, t \in T \tag{4.8}$$

donde T es el conjunto de todos los períodos de tiempo considerados.

4.8.3. Producción por celda en cada periodo

La producción de cada parcela en cada período de tiempo, $P_{s,t}$, se calcula como el producto de la productividad por hectárea, r_s, y el área de la parcela, $A_{s,t}$. Para períodos de tiempo posteriores al primero, se considera la tasa de crecimiento, g:

$$P_{s,1} = r_s \times A_{s,1} \tag{4.9}$$

$$P_{s,t} = P_{s,t-1} \times (1+g) \quad \forall s \in S, t \in T \backslash \{1\} \tag{4.10}$$

4.8.4. Costo de cada celda en cada periodo de tiempo

El costo de cada parcela en cada período de tiempo, $C_{s,t}$, se calcula como la suma del costo de cosecha por volumen de madera, c_{vol}, multiplicado por la producción de la parcela, $P_{s,t}$, y el costo de cosecha por área, $c_{\text{área}}$, multiplicado por el área de la parcela, $A_{s,t}$, más el costo fijo, c_{fijo}. Para períodos de tiempo posteriores al primero, se considera la tasa de descuento, d:

$$C_{s,1} = P_{s,1} \times c_{\text{vol}} + A_{s,1} \times c_{\text{área}} + c_{\text{fijo}} \tag{4.11}$$

$$C_{s,t} = C_{s,t-1} \times (1-d) \quad \forall s \in S, t \in T \backslash \{1\} \tag{4.12}$$

4.8.5. Cotas de demanda

La demanda mínima y máxima de madera en cada período de tiempo, $D_{\text{min},t}$ y $D_{\text{max},t}$, se calculan en función de la demanda mínima y máxima relativas, dem_{min} y dem_{max}, y la producción total en el primer período de tiempo, $\sum_{s \in S} P_{s,1}$, repartida entre los períodos de tiempo, $|T|$. Para períodos de tiempo posteriores al primero, se considera la tasa de crecimiento, g:

$$D_{\text{min},1} = \text{dem}_{\text{min}} \times \frac{\sum_{s \in S} P_{s,1}}{|T|} \tag{4.13}$$

$$D_{\text{max},1} = \text{dem}_{\text{max}} \times \frac{\sum_{s \in S} P_{s,1}}{|T|} \tag{4.14}$$

$$D_{\text{min},t} = D_{\text{min},t-1} \times (1+g) \quad \forall t \in T \backslash \{1\} \tag{4.15}$$

$$D_{\text{max},t} = D_{\text{max},t-1} \times (1+g) \quad \forall t \in T \backslash \{1\} \tag{4.16}$$

La definición de estas cotas demandas en base a la producción permite generar un escenario base de la evolución de dichas cotas, este es utilizado para obtener los parámetros base μ y σ que serán utilizados para generar el árbol de escenarios de la demanda.

Capítulo 5

Experimentos e implementación

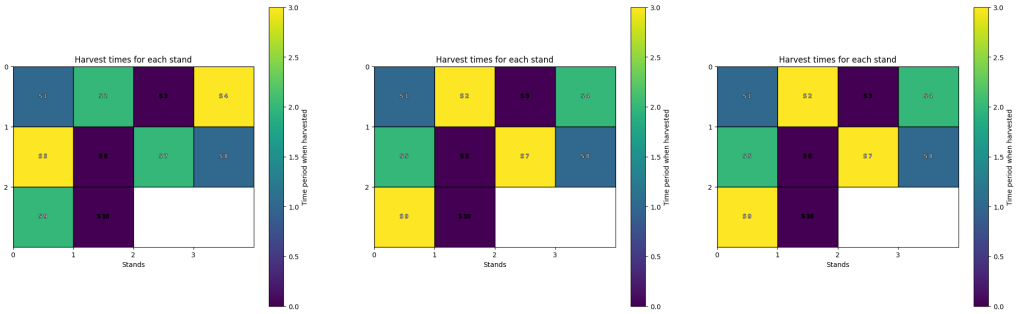
5.1. Solución base inicial: solución independiente de escenarios

Como primera acercamiento a la solución para los parámetros del bosque definidos en la sección () se considera la solución independiente de cada escenario. En base a esto se tienen los siguiente resultados:

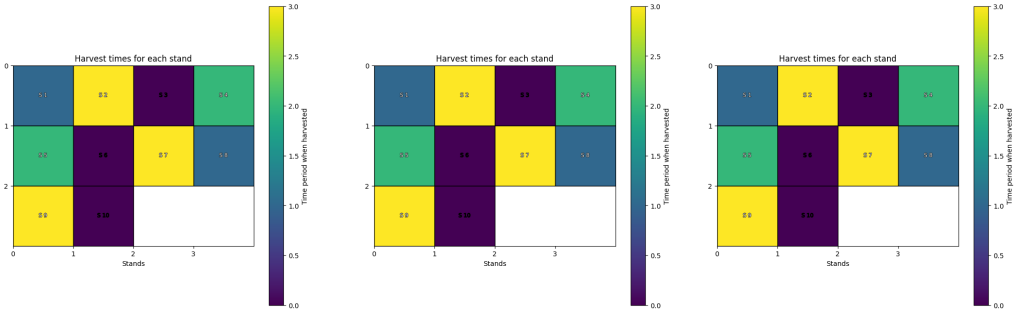
Tabla 5.1: 10 stands: Valores de la función objetivo para diferentes escenarios.

Escenario	**Función Objetivo**
1	5,600,287
2	6,051,848
3	6,401,898
4	4,390,910
5	4,454,242
6	4,555,572
7	3,794,396
8	3,849,091
9	3,240,388
$\bar{Z}$	**4,711,714**

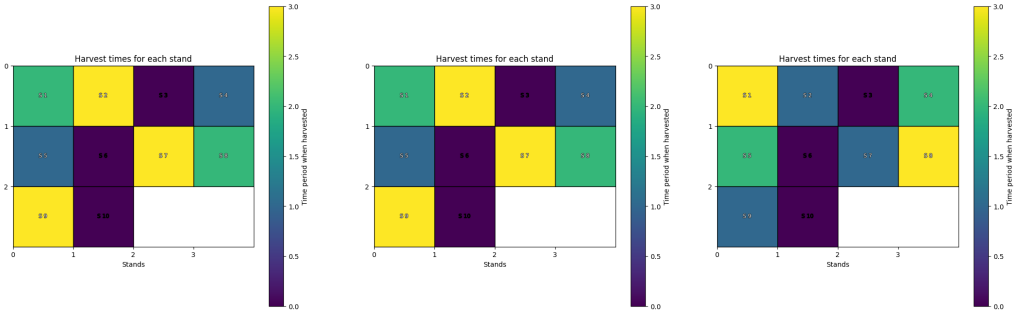
A continuación se puede ver para cada escenario cual es la acción óptima según el periodo de tiempo:



(a) Escenario 1. (b) Escenario 2. (c) Escenario 3.



(d) Escenario 4. (e) Escenario 5. (f) Escenario 6.

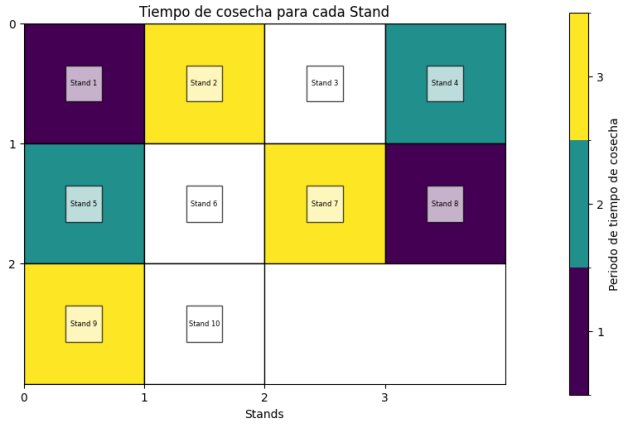


(g) Escenario 7. (h) Escenario 8. (i) Escenario 9.

Figura 5.1: 10 stands. Solución para cada escenario resuelto de manera independiente

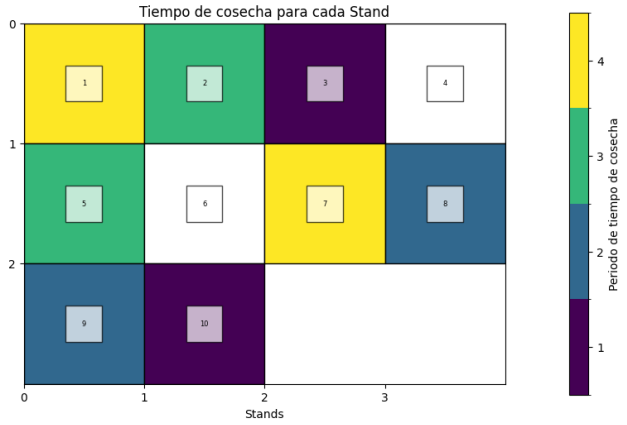
5.2. Resolución del problema

5.2.1. Solución óptima: 10 stands - 9 Escenarios



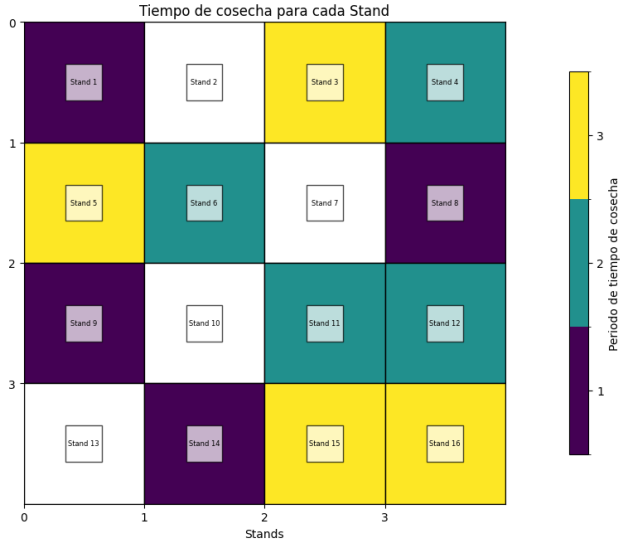
(a) PH - 10 stands

5.2.2. Solución óptima: 10 stands - 27 Escenarios



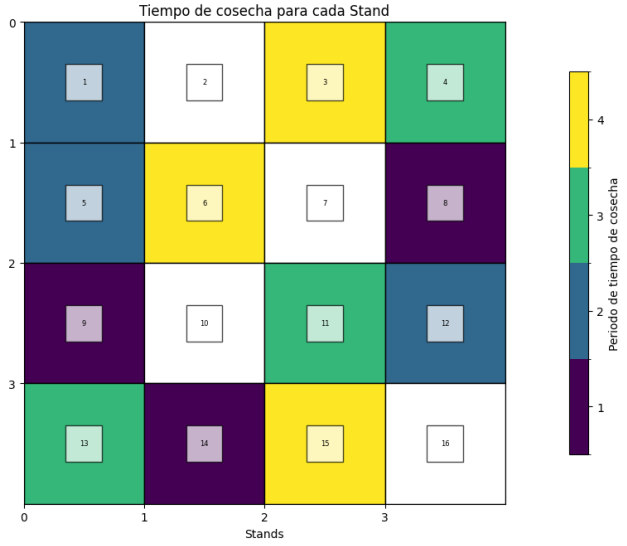
(a) PH - 10 stands - 27 escenarios

5.2.3. Solución óptima: 16 stands - 9 Escenarios



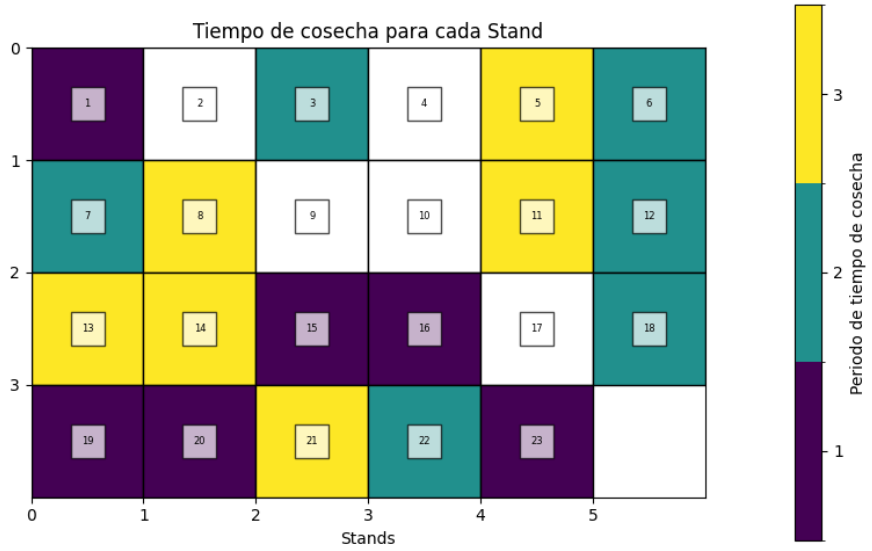
(a) PH - 16 stands

5.2.4. Solución 16 stands - 27 Escenarios



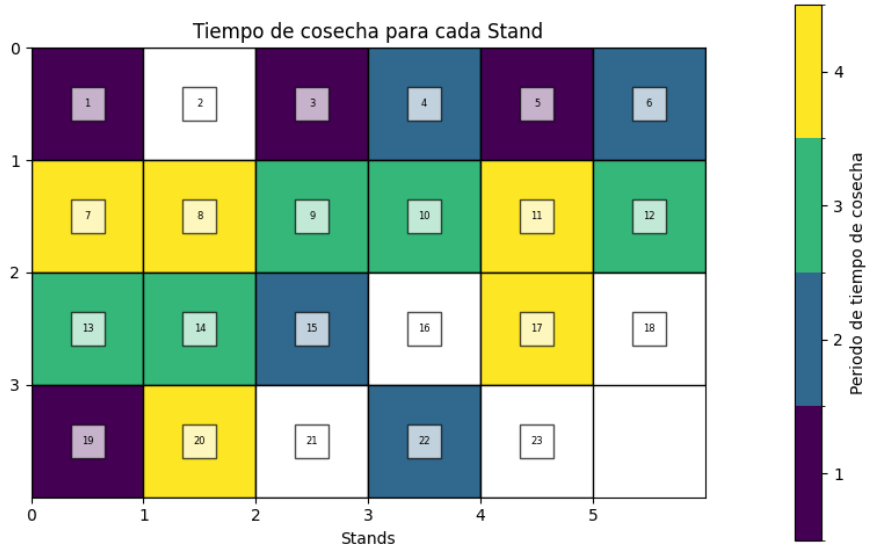
(a) PH - 16 stands - 27 Escenarios

5.2.5. Solución óptima: 23 stands - 9 escenarios



(a) PH - 23 stands

5.2.6. Solución : 23 stands - 27 escenarios



(a) PH - 23 stands

5.2.7. Incorporación de Penalizaciones en la Función de Recompensa

En la formulación de la función de recompensa se incorporan elementos de penalización que buscan disuadir al agente de ejecutar acciones que son inviables, las cuales para este caso

vienen dadas por la violación de las restricciones definidas en el problema.

La inclusión de penalizaciones en la función de recompensa puede acelerar la convergencia del algoritmo al evitar que el agente explore estados inviables. Por otra parte, las penalizaciones funcionan como mecanismos correctivos que alinean la política del agente más estrechamente con las metas de optimización a largo plazo, sobreponiéndose a las limitaciones inherentes a la retroalimentación inmediata. En este caso, dado que se ha incorporado una máscara que evita que el agente seleccione celdas previamente cosechadas, la penalización viene dada para el caso en el cual el agente decide cosechar fuera de los límites de demanda establecidos.

5.2.8. Interacción entre Estocasticidad y la Función de Recompensa

Es crucial reconocer que la estocasticidad en los precios y la demanda de madera puede limitar la eficacia de la función de recompensa para guiar al agente hacia decisiones óptimas. Dado que la función de recompensa se basa en información disponible en un momento dado, su capacidad para reflejar el valor verdadero de una acción en un ambiente estocástico puede ser limitada. Específicamente, una acción que parece óptima en el contexto de la información actual puede resultar ser subóptima cuando se toma en cuenta la variabilidad en los precios y la demanda.

5.3. Parámetros del experimento

Para la optimización de hiperparámetros en experimentos de Reinforcement Learning, se adoptaron dos estrategias principales: la búsqueda aleatoria (Random Search) y el uso de Optuna, un marco de trabajo avanzado para optimización automática.

Inicialización para la Política y las Funciones de Valor

La correcta inicialización de la política y las funciones de valor es fundamental, ya que puede influir significativamente en la velocidad de convergencia y la calidad de la política óptima aprendida. Se optó por una inicialización aleatoria de los pesos de las Redes Neuronales en cada experimento.

Efectos de la Inicialización en el Balance entre Exploración y Explotación

Una inicialización inapropiada puede desequilibrar la relación entre exploración y explotación. Se empleó una estrategia ϵ-greedy, iniciando con un valor elevado de ϵ, favoreciendo la exploración al principio, y disminuyéndolo progresivamente durante el entrenamiento.

Parámetros del Algoritmo y Estrategias de Optimización

La búsqueda aleatoria se llevó a cabo mediante una función que selecciona aleatoriamente combinaciones de hiperparámetros de un conjunto predefinido. Los hiperparámetros incluyen la capacidad del buffer, valores iniciales y finales de ϵ, tasa de decaimiento de ϵ, número de episodios, tamaño del lote, factor de descuento γ, intervalo de actualización de la red objetivo y tasa de aprendizaje.

Paralelamente, se utilizó Optuna para una optimización dirigida y eficiente. Optuna automatiza la selección de hiperparámetros mediante la definición de una función objetivo y un espacio de búsqueda, donde técnicas como TPE o CMA-ES guían la selección de hiperparámetros en función de los resultados de iteraciones anteriores, mejorando la eficiencia del proceso de búsqueda.

Evaluación de Desempeño

El rendimiento se evaluó utilizando un enfoque totalmente "greedy"después de cada iteración de entrenamiento con un conjunto específico de hiperparámetros, midiendo la recompensa media obtenida durante un número fijo de episodios de evaluación.

5.4. Tabla de Búsqueda de Hiperparámetros

Tabla 5.2: Rangos de Muestreo para la Búsqueda de Hiperparámetros

Parámetro	Rango de Muestreo
Capacidad del Buffer	$[10,000, 60,000]$
$\epsilon_{Inicial}$	$[0.5, 1.0]$
ϵ_{Final}	$[0.001, 0.1]$
Tasa de Decaimiento de ϵ	$[0.0001, 0.001]$
Número de Episodios	$[10,000, 20,000]$
Tamaño del Lote	$[64, 128, 256, 512]$
γ	$[0.95, 0.99]$
Target update	$[5, 30]$
Tasa de Aprendizaje	$[0.001, 0.01]$

Ambas metodologías, Random Search y el uso de Optuna, se aplicaron para identificar los hiperparámetros más efectivos dentro de un rango viable. La eficacia de cada conjunto de hiperparámetros se evaluó por su rendimiento en episodios específicos, registrando los de mejor rendimiento para su uso en futuros entrenamientos. Este proceso se repitió varias veces para asegurar la consistencia del rendimiento observado, obteniendo así una curva promedio de recompensa y su desviación estándar.

Capítulo 6

Resultados y análisis

6.1. Curvas de Aprendizaje

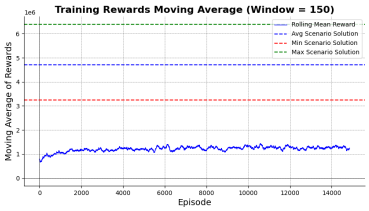
Dentro de este capítulo, se presentan los resultados obtenidos de los distintos algoritmos aplicados al problema. Las curvas de aprendizaje, que muestran la evolución de la función de recompensa a lo largo de los episodios de entrenamiento, son herramientas clave para evaluar la efectividad y eficiencia de estos algoritmos en el contexto del problema.

En las secciones siguientes, se mostrarán gráficos representativos de las curvas de aprendizaje para diferentes instancias del problema, diferenciadas por el número de "standsz el algoritmo utilizado.

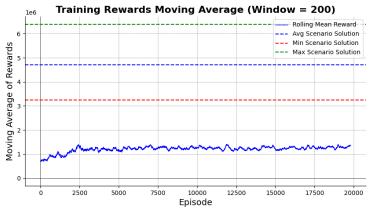
Para cada variante del problema, se analizaron tanto ambientes restringidos, donde las decisiones están sujetas a ciertas restricciones relacionadas demandas, como ambientes irrestrictos, donde estas restricciones no aplican. Esta aproximación dual nos permite entender la capacidad de los algoritmos para cumplir con restricciones operativas, y al mismo tiempo, examinar su comportamiento en escenarios más amplios, brindando una visión completa de su aplicabilidad y robustez.

6.1.1. Ambiente 1: Árbol de precios y demandas

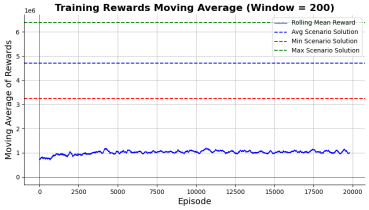
DQN - 10 stands



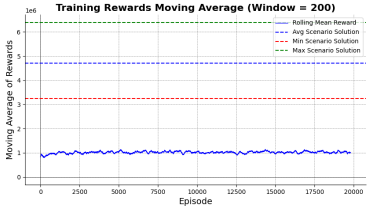
(a) DQN 1.



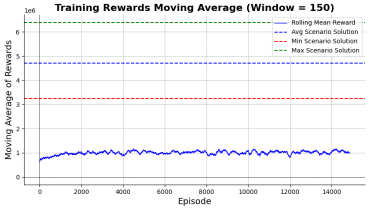
(b) DQN 2.



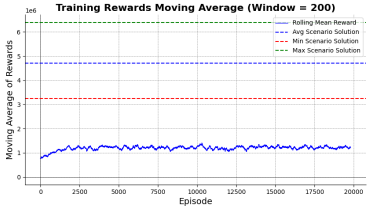
(c) DQN 3.



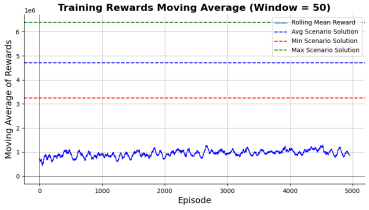
(d) DQN 4.



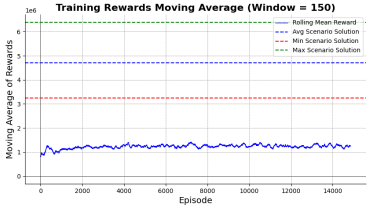
(e) DQN 5.



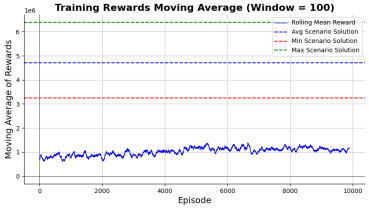
(f) DQN 6.



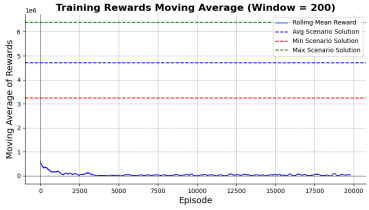
(g) DQN 7.



(h) DQN 8.



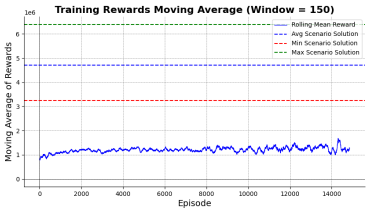
(i) DQN 9.



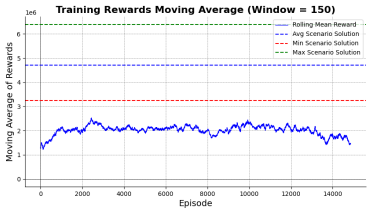
(j) DQN 10.

Figura 6.1: Curvas de entrenamiento para DQN ambiente con demanda restringida. Búsqueda aleatoria de hiperparámetros

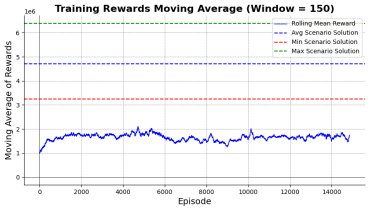
Dueling DQN - NN arch 1 - 10 stands



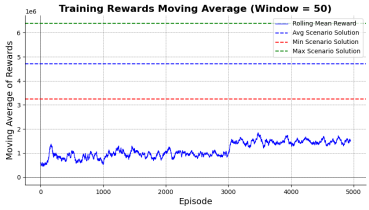
(a) Dueling DQN 1.



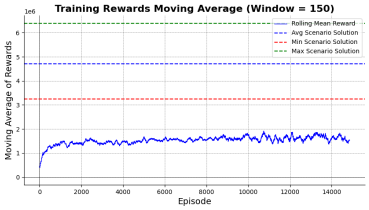
(b) Dueling DQN 2.



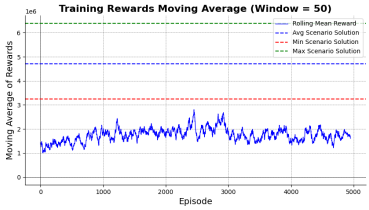
(c) Dueling DQN 3.



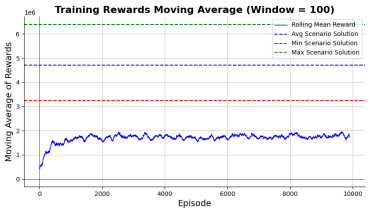
(d) Dueling DQN 4.



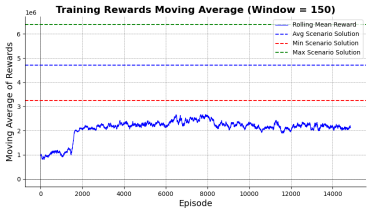
(e) Dueling DQN 5.



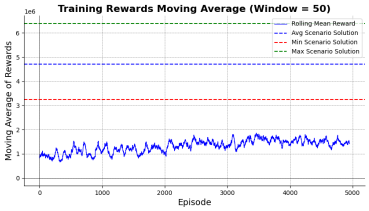
(f) Dueling DQN 6.



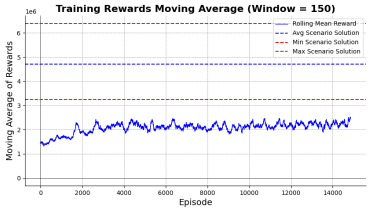
(g) Dueling DQN 7.



(h) Dueling DQN 8.

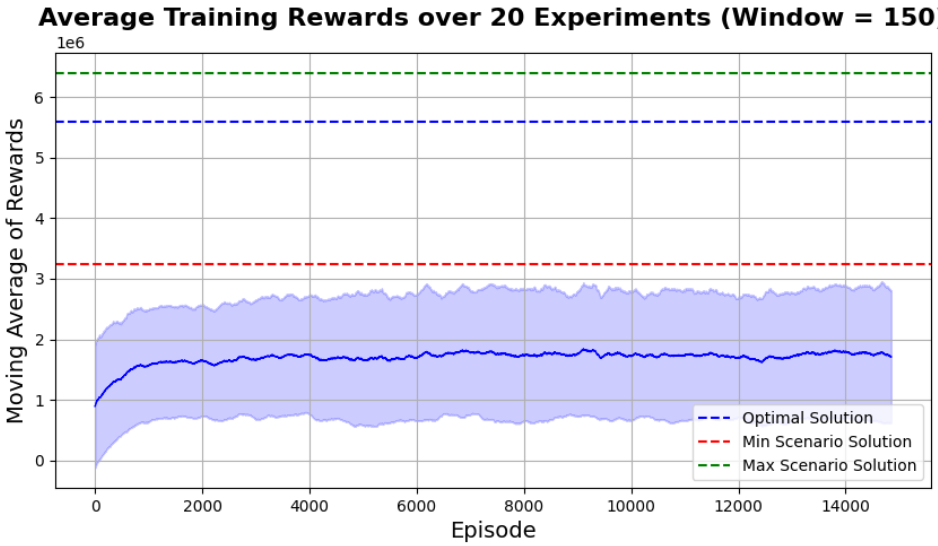


(i) Dueling DQN 9.



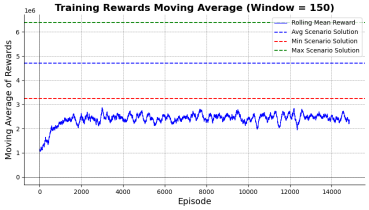
(j) Dueling DQN 10.

Figura 6.2: Training Curves for Dueling DQN in an Environment with Restricted Demand. Random Hyperparameter Search

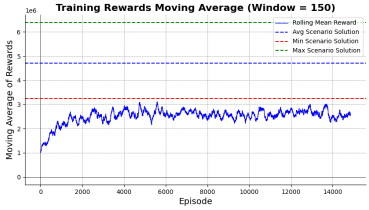


(a) Curva promedio para 20 entrenamientos utilizando hiperparámetros con mejor rendimiento.

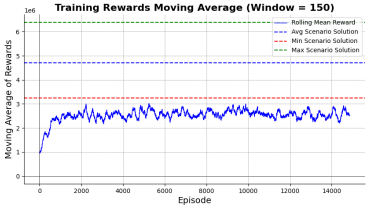
Dueling DQN - NN arch 2 - 10 stands



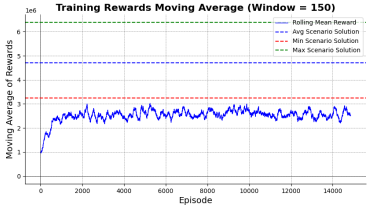
(a) Dueling DQN Arch 2 1.



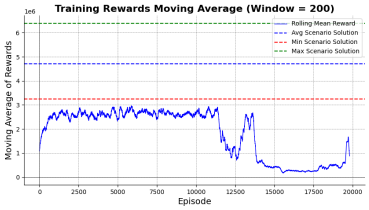
(b) Dueling DQN Arch 2 2.



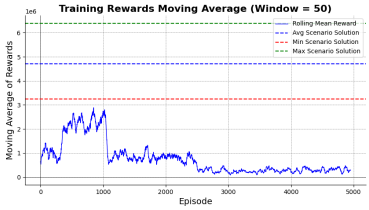
(c) Dueling DQN Arch 2 3.



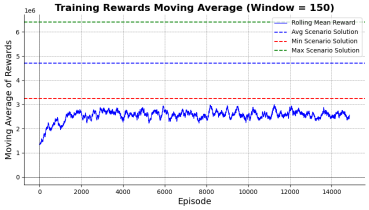
(d) Dueling DQN Arch 2 4.



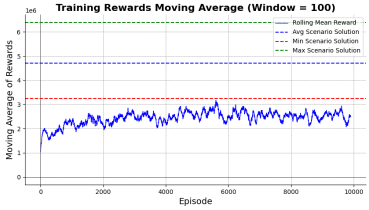
(e) Dueling DQN Arch 2 5.



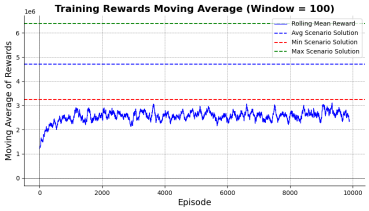
(f) Dueling DQN Arch 2 6.



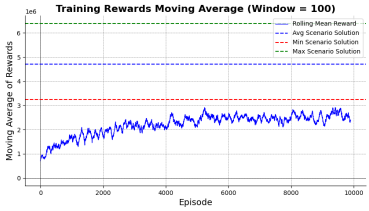
(g) Dueling DQN Arch 2 7.



(h) Dueling DQN Arch 2 8.

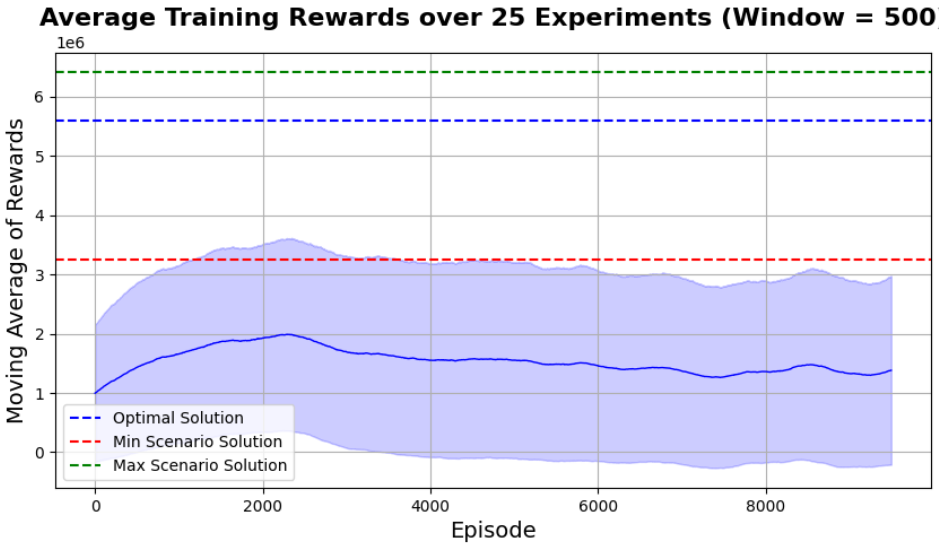


(i) Dueling DQN Arch 2 9.



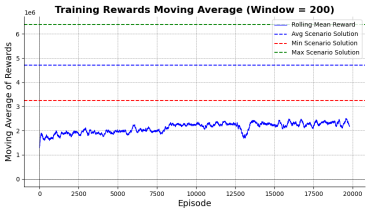
(j) Dueling DQN Arch 2 10.

Figura 6.4: Training Curves for Dueling DQN with NN new architecture in an Environment with Restricted Demand. Random Hyperparameter Search

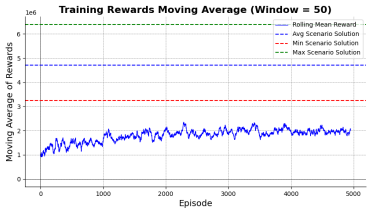


(a) Curva promedio para 25 entrenamientos utilizando hiperparámetros con mejor rendimiento.

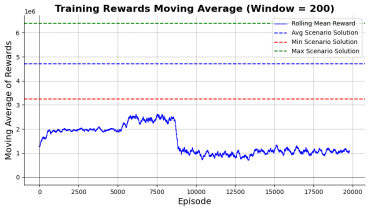
Dueling DQN - NN arch 3 - 10 stands



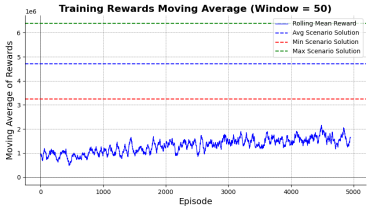
(a) Dueling DQN Arch 3 1.



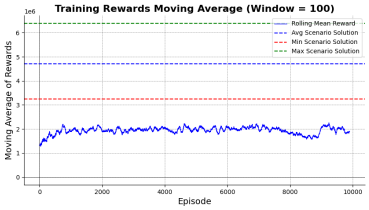
(b) Dueling DQN Arch 3 2.



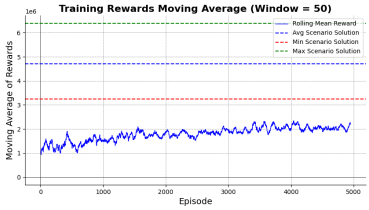
(c) Dueling DQN Arch 3 3.



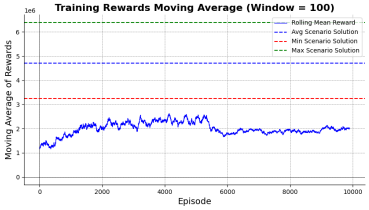
(d) Dueling DQN Arch 3 4.



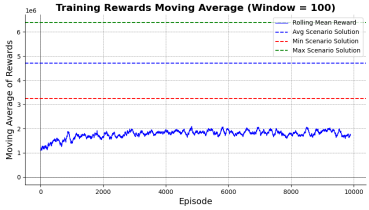
(e) Dueling DQN Arch 3 5.



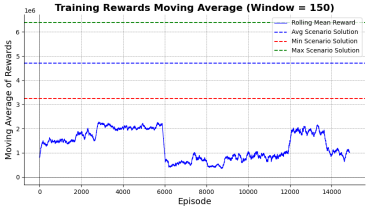
(f) Dueling DQN Arch 3 6.



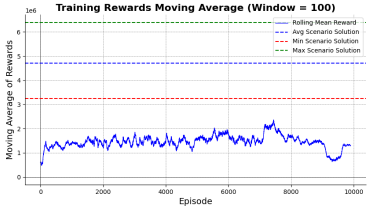
(g) Dueling DQN Arch 3 7.



(h) Dueling DQN Arch 3 8.

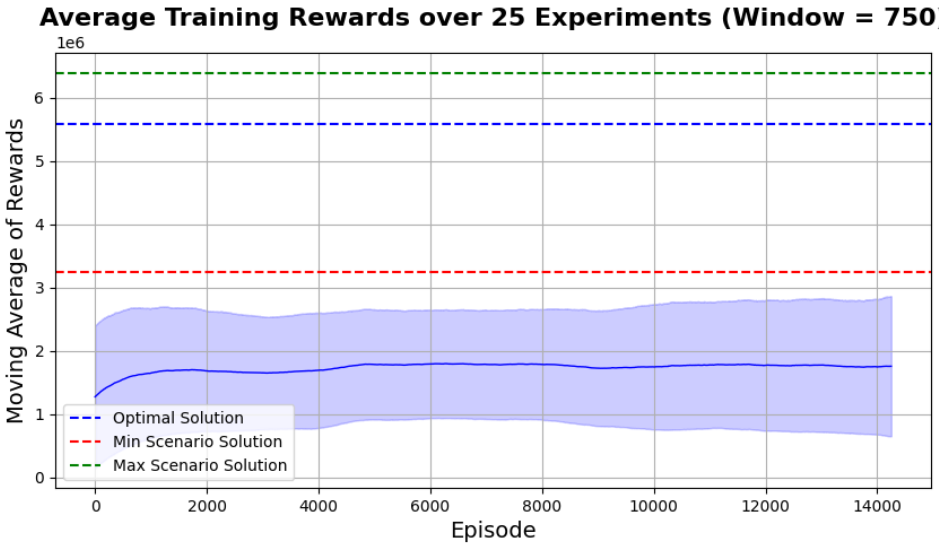


(i) Dueling DQN Arch 3 9.



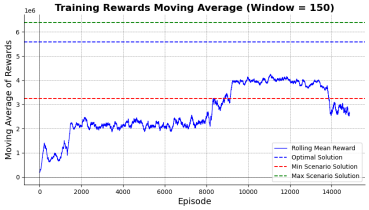
(j) Dueling DQN Arch 3 10.

Figura 6.6: Training Curves for Dueling DQN with NN new architecture in an Environment with Restricted Demand. Random Hyperparameter Search

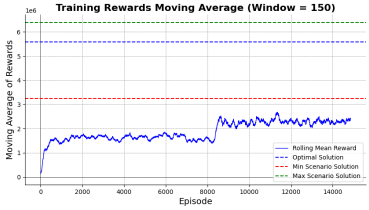


(a) Curva promedio para 25 entrenamientos utilizando hiperparámetros con mejor rendimiento.

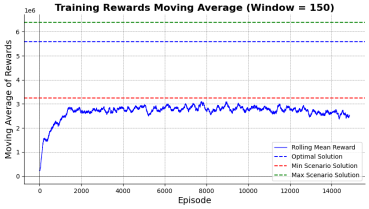
Double Dueling DQN - 10 stands



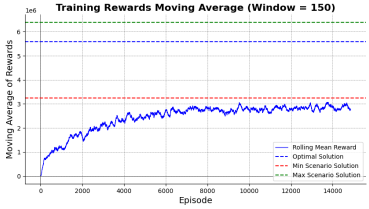
(a) Double Dueling DQN 1.



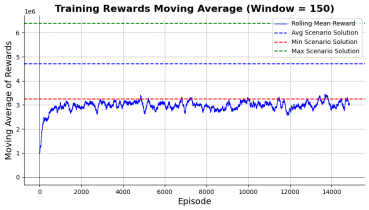
(b) Double Dueling DQN 2.



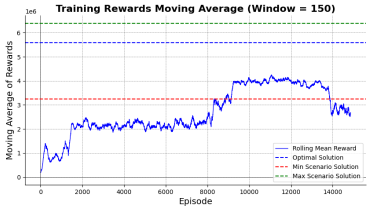
(c) Double Dueling DQN 3.



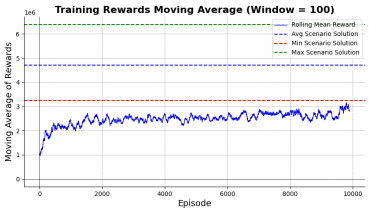
(d) Double Dueling DQN 4.



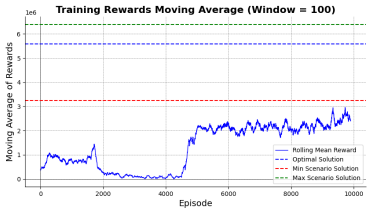
(e) Double Dueling DQN 5.



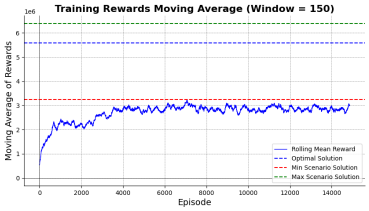
(f) Double Dueling DQN 6.



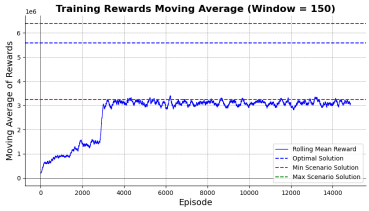
(g) Double Dueling DQN 7.



(h) Double Dueling DQN 8.

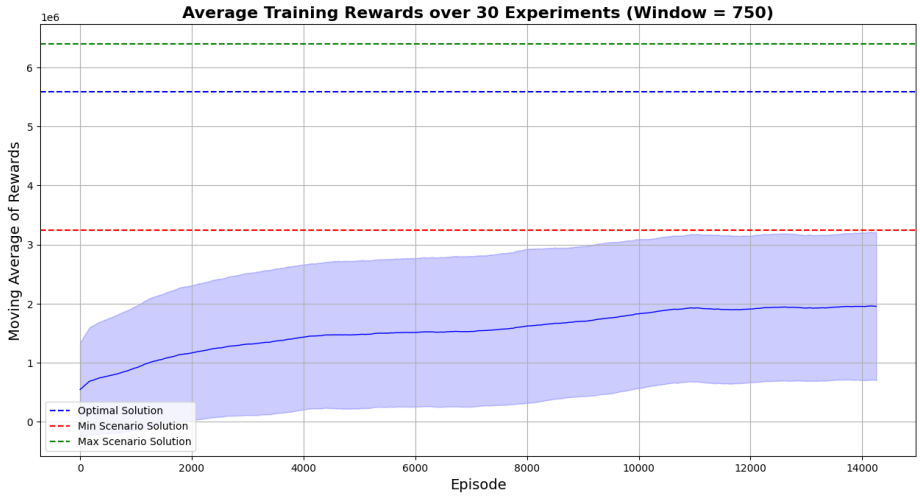


(i) Double Dueling DQN 9.

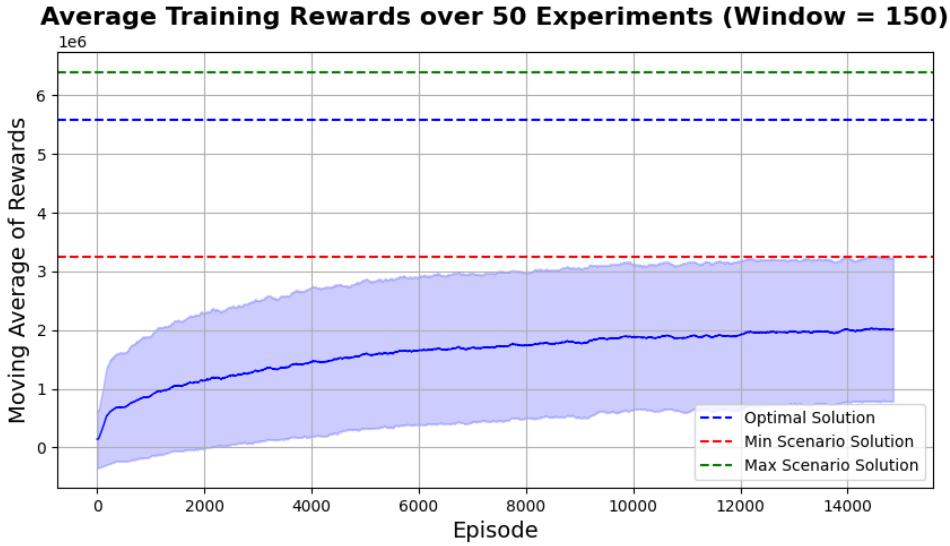


(j) Double Dueling DQN 10.

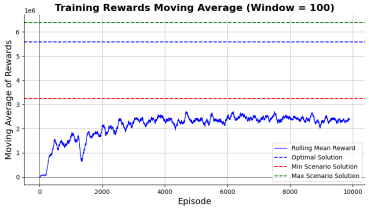
Figura 6.8: Training Curves for Double Dueling DQN with NN new architecture in an Environment with Restricted Demand. Random Hyperparameter Search



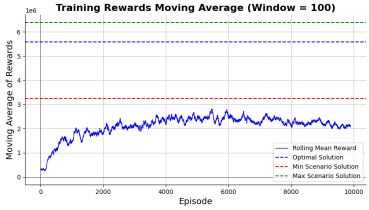
(a) Curva promedio para 30 entrenamientos utilizando hiperparámetros con mejor rendimiento.



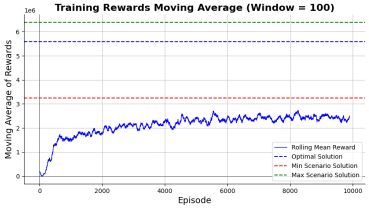
(a) Curva promedio para 50 entrenamientos utilizando hiperparámetros con mejor rendimiento.



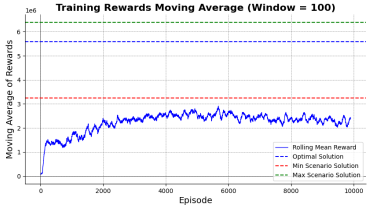
(a) Double Dueling DQN with PER 1.



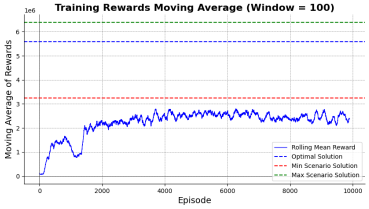
(b) Double Dueling DQN with PER 2.



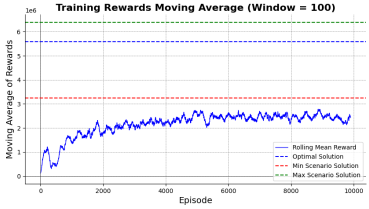
(c) Double Dueling DQN with PER 3.



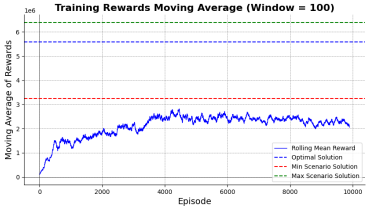
(d) Double Dueling DQN with PER 4.



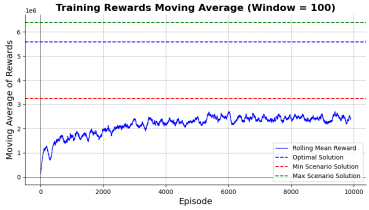
(e) Double Dueling DQN with PER 5.



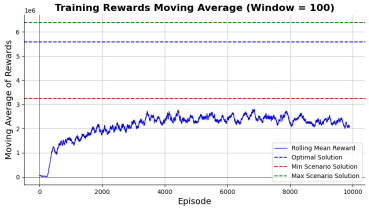
(f) Double Dueling DQN with PER 6.



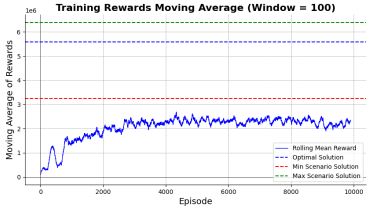
(g) Double Dueling DQN with PER 7.



(h) Double Dueling DQN with PER 8.

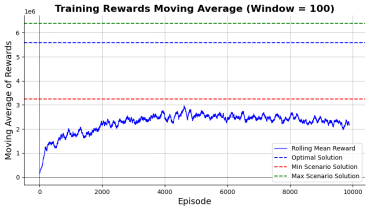


(i) Double Dueling DQN with PER 9.

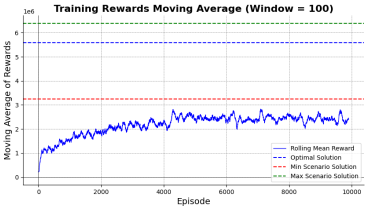


(j) Double Dueling DQN with PER 10.

Figura 6.11: Training Curves for Double Dueling DQN with Prioritized Experience Replay in an Environment with Restricted Demand. Optimal parameters based on Optuna



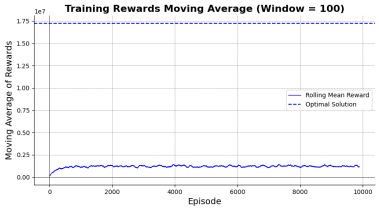
(a) Double Dueling DQN 11.



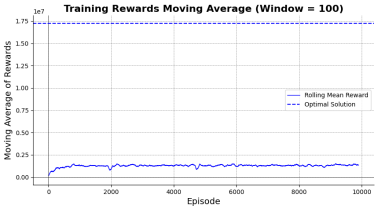
(b) Double Dueling DQN 12.

Figura 6.12: Training Curves for Double Dueling DQN with Prioritized Experience Replay in an Environment with Restricted Demand. Optimal parameters based on Optuna

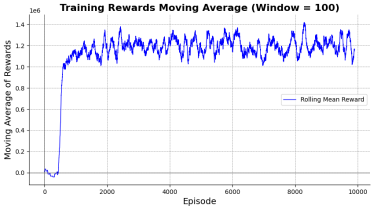
Double Dueling DQN Prioritized Experience Replay - 10 stands - 27 Scenarios



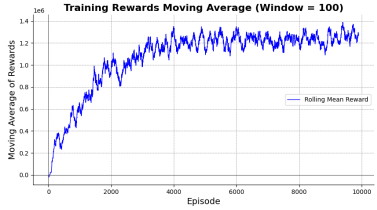
(a) Double Dueling DQN - PER - 27 Scenarios 1.



(b) Double Dueling DQN - PER - 27 Scenarios 2.



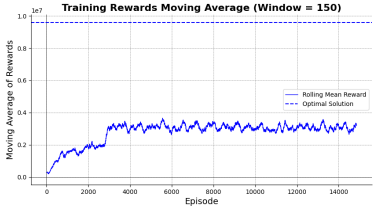
(c) Double Dueling DQN with PER 27 Scenarios 1.



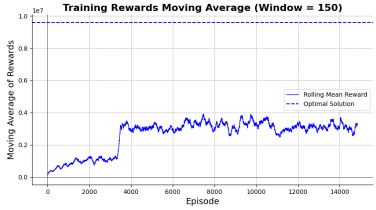
(d) Double Dueling DQN with PER 27 Scenarios 2.

Figura 6.13: Training Curves for Double Dueling DQN with Prioritized Experience Replay in an Environment with Restricted Demand. 27 scenarios

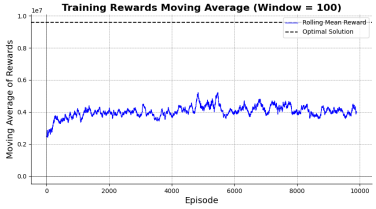
Double Dueling DQN - 16 stands



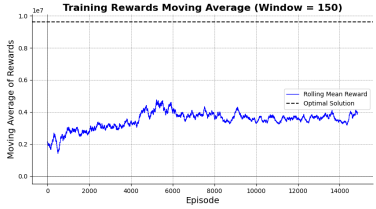
(a) Double Dueling DQN 1.



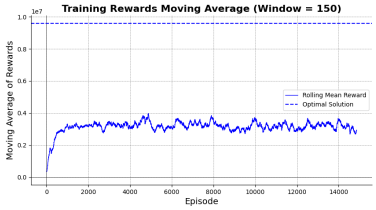
(b) Double Dueling DQN 2.



(c) Double Dueling DQN 3.

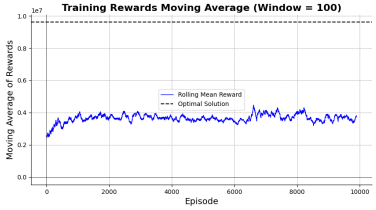


(d) Double Dueling DQN 4.

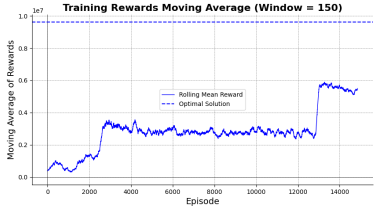


(e) Double Dueling DQN 5.

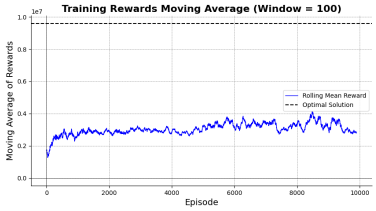
(f) Double Dueling DQN 6.



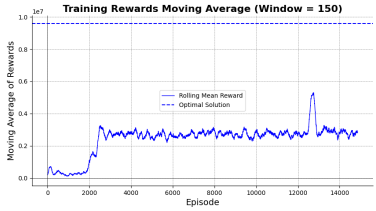
(g) Double Dueling DQN 7.



(h) Double Dueling DQN 8.



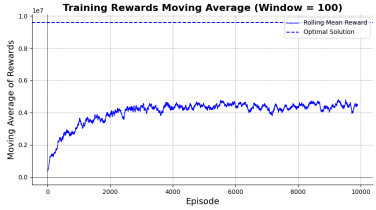
(i) Double Dueling DQN 9.



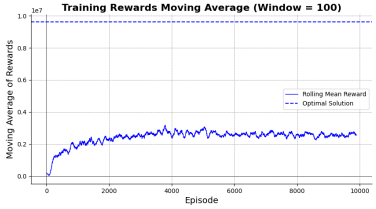
(j) Double Dueling DQN 10.

Figura 6.14: Training Curves for Double Dueling DQN with NN new architecture in an Environment with Restricted Demand. Random Hyperparameter Search

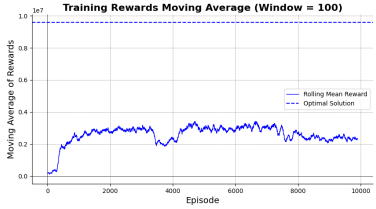
(a) Double Dueling DQN with PER 1.



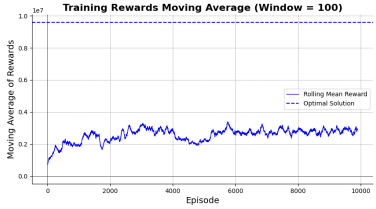
(b) Double Dueling DQN with PER 2.



(c) Double Dueling DQN with PER 3.



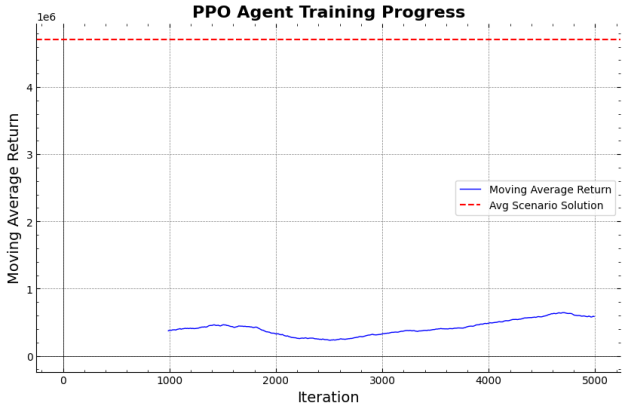
(d) Double Dueling DQN with PER 4.



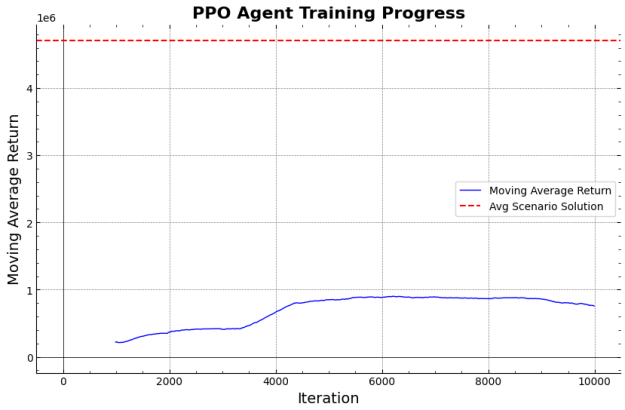
(e) Double Dueling DQN with PER 3.

Figura 6.15: Training Curves for Double Dueling DQN with Prioritized Experience Replay in an Environment with Restricted Demand. Optimal parameters based on Optuna - 16 stands 9 scenarios

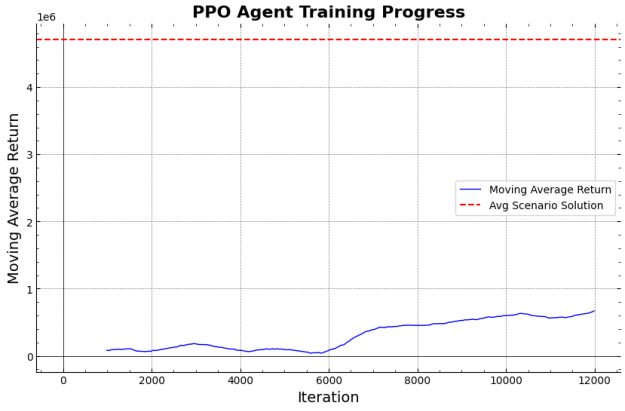
PPO - 10 stands



(a) PPO 5000 episodios.



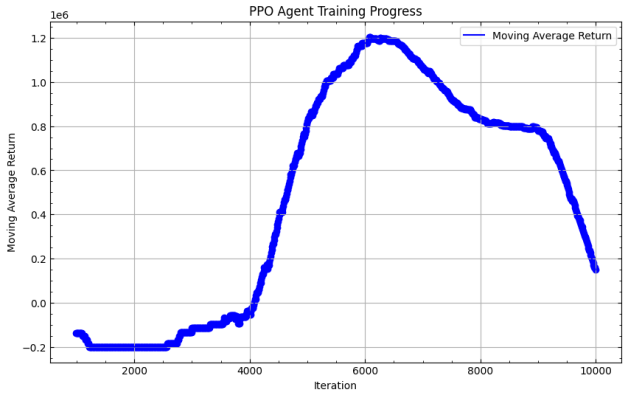
(b) PPO 10000 episodios.



(c) PPO 12000 episodios.

Figura 6.16: PPO para ambiente con escenario de precios y demandas restringidas

PPO 0.3 clipping más capas

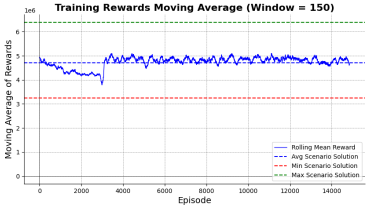


(a) PPO 10000 episodios.

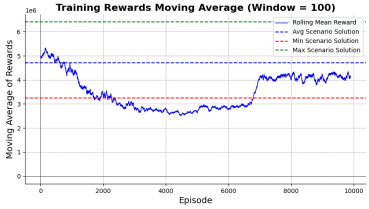
6.1.2. Ambiente 2: Árbol de precios y demanda irrestricta

DQN

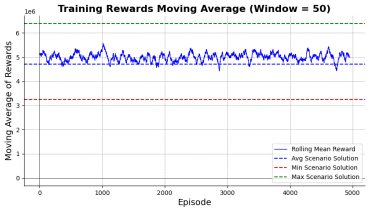
Dueling DQN



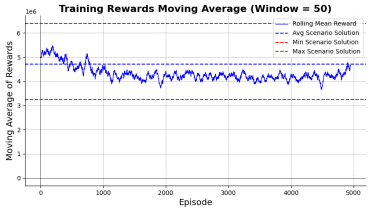
(a) Dueling DQN 1.



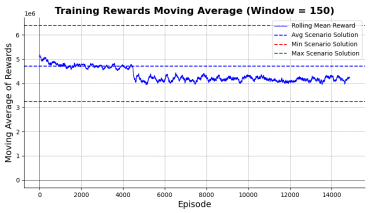
(b) Dueling DQN 2.



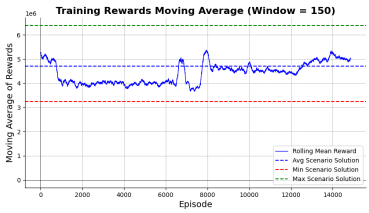
(c) Dueling DQN 3.



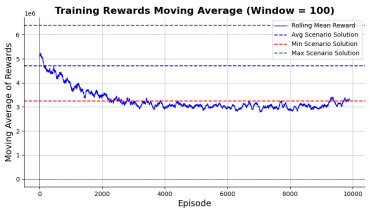
(d) Dueling DQN 4.



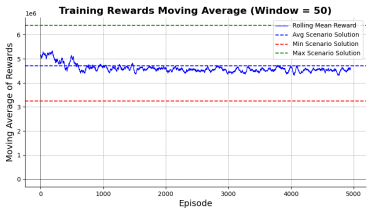
(e) Dueling DQN 5.



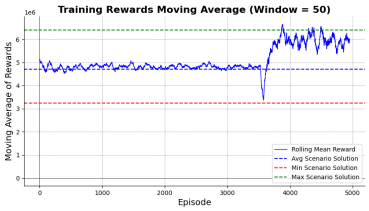
(f) Dueling DQN 6.



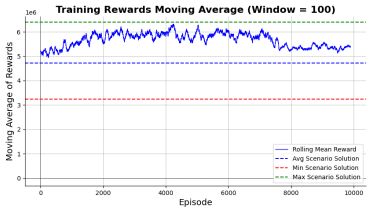
(g) Dueling DQN 7.



(h) Dueling DQN 8.



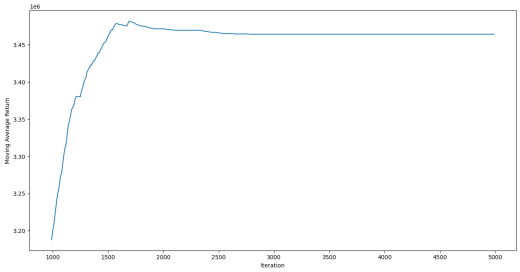
(i) Dueling DQN 9.



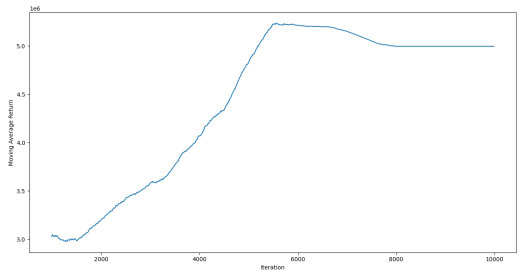
(j) Dueling DQN 10.

Figura 6.18: Curvas de entrenamiento para Dueling DQN sobre ambiente con demanda irrestricta. Búsqueda aleatoria de hiperparámetros

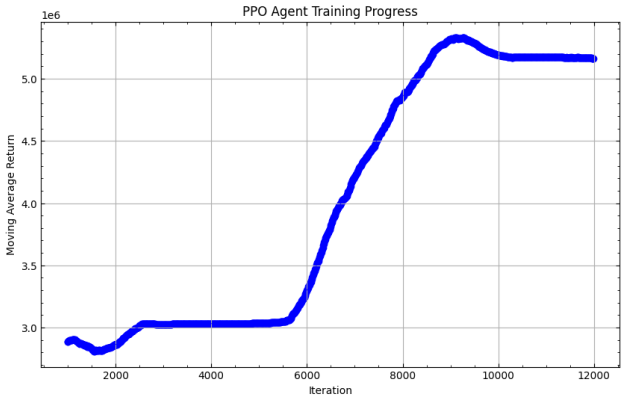
PPO - 10 stands



(a) PPO 5000 episodios.



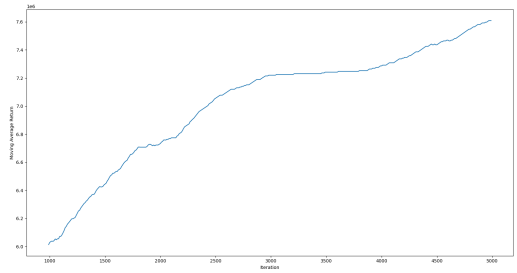
(b) PPO 10000 episodios.



(c) PPO 12000 episodios.

Figura 6.19: 10 stands: PPO para ambiente con escenario de precios y demandas irrestrictas

PPO - 23 stands



(a) PPO 5000 episodios.

Capítulo 7

Discusión

7.1. Análisis de curvas recompensas acumuladas

7.1.1. Ambiente 1: Cotas de demanda

Modelo de 10 stands - 9 escenarios

Para abordar el problema de optimización, se tomó una versión simplificada inicial, centrada en 10 stands de cosecha a lo largo de un horizonte temporal de 3 años. A pesar de las dimensiones manejables de esta instancia, se incluyen 9 escenarios distintos, añadiendo una complejidad intrínseca. Este ambiente fusiona un árbol de escenarios de precios con las respectivas cotas de demanda. Estas últimas generan restricciones adicionales que, al enfrentar una política de decisión inicialmente aleatoria durante las primeras etapas del entrenamiento, inducen penalizaciones recurrentes. Es esperado que a medida que las redes neuronales realicen ajustes y perfeccionen sus parámetros, la frecuencia y magnitud de estas penalizaciones se reduzcan y las decisiones converjan hacia una solución óptima.

En el proceso de entrenamiento, se realizó un extenso análisis experimental: aproximadamente 50 entrenamientos por cada algoritmo utilizando random search para probar los diferentes hiperparámetros, execptuando el caso del algoritmo PPO, cuyo tiempo de entrenamiento fue considerablemente mayor. No obstante, para propósitos de claridad y concisión en este análisis, sólo se presentan aquellos resultados que exhiben las mejores curvas de rendimiento. La selección de estas curvas se basó en criterios cualitativos, priorizando la convergencia y estabilidad de la recompensa acumulada, cuya curva se obtiene como un promedio móvil para el 1 % de los episodios.

Al someter a prueba varios algoritmos y sus respectivas variantes en el ambiente estocástico definido, se desvelaron patrones distintivos en su rendimiento:

- **DQN**: Su rendimiento fue insatisfactorio. No se distingue una convergencia clara y las recompensas oscilan frecuentemente en torno al millón. Si bien en algunos experimentos se percibe un ligero incremento en la recompensa, en otros se evidencia una inclinación hacia el mínimo. La variabilidad en las recompensas es pronunciada y, dada esta inconsistencia, no se procedió a optimizar los parámetros.
- **Dueling DQN Arquitectura 1**: Aunque se detecta una mejora marginal en ciertos experimentos, donde las recompensas superan los 2 millones, persiste una variabilidad

notable. La ausencia de convergencia es palpable en numerosos experimentos. Un análisis de 20 experimentos con óptimos hiperparámetros sugiere una tendencia ascendente en las etapas iniciales que eventualmente se estabiliza. Sin embargo, la variabilidad, denotada por la desviación estándar, indica que la solución óptima sigue siendo esquiva.

- **Dueling DQN Arquitectura 2**: Al modificar la arquitectura de la red e incorporar la normalización por lotes, se aprecian indicios de convergencia en las fases iniciales de ciertos experimentos. No obstante, algunos experimentos revelan comportamientos atípicos, con recompensas que se desploman tras alcanzar máximos locales. Un análisis con los hiperparámetros de mejor rendimiento, a lo largo de 25 episodios, indica un incremento inicial seguido de una ligera reducción. La amplia banda de desviación estándar, no obstante, sugiere incertidumbre en las estimaciones, lo que requiere una exploración más profunda de las causas subyacentes.

- **Dueling DQN Arquitectura 3**: Aunque los resultados son paralelos al modelo anterior, la estabilidad parece ser ligeramente superior, con una desviación estándar más contenida. Se observa una moderada convergencia entre los episodios 4,000 y 15,000.

- **Double Dueling DQN**: Este algoritmo presenta la performance más destacada. En numerosos experimentos con diferentes conjuntos de hiperparámetros, se nota una tendencia creciente que eventualmente converge y estabiliza. Las recompensas, en ciertas ocasiones, se aproximan o incluso superan brevemente la solución independiente del peor escenario. Estas fluctuaciones pueden atribuirse a la variabilidad inherente en el muestreo de precios al interactuar con el ambiente.

- **Double Dueling DQN w/ PER**: El análisis de las recompensas acumuladas revela que, aunque el algoritmo muestra una progresión positiva en el aprendizaje, aún no logra converger hacia la solución óptima. El uso de PER no parece alcanzar la eficacia esperada, ya que la recompensa promedio no solo no supera la solución del escenario mínimo, sino que también se mantiene considerablemente por debajo de la solución óptima y de los escenarios máximos posibles. Estos resultados sugieren que la estrategia de priorización de experiencias y la configuración general del algoritmo requieren una revisión y ajuste significativos para mejorar el rendimiento y la convergencia del modelo. Hay un margen considerable para la optimización que podría explorarse mediante el ajuste de hiperparámetros, la mejora de la arquitectura de la red neuronal o la implementación de una función de recompensa más efectiva.

    1. **Convergencia y Estabilidad**: La línea azul sólida, que representa la recompensa promedio móvil de entrenamiento, muestra un incremento durante los episodios, reflejando que el algoritmo está aprendiendo y mejorando su desempeño con el tiempo. Sin embargo, la línea no muestra una convergencia sólida hacia la línea punteada azul que representa la solución óptima, indicando que aún hay margen para mejorar la configuración del algoritmo para alcanzar una política de decisiones óptima.
    2. **Impacto del Prioritized Experience Replay**: A pesar de que el PER está diseñado para acelerar el aprendizaje al enfocarse en las experiencias más valiosas, la falta de convergencia hacia la solución óptima sugiere que la técnica de priorización no está siendo tan efectiva como se espera. Esto podría significar que la metodología

actual de priorización de experiencias necesita ajustes para maximizar su potencial de aprendizaje en este entorno.

3. **Comparación con Escenarios Extremos**: La recompensa promedio no supera el rendimiento de la solución del escenario mínimo, representada por la línea roja punteada, lo que indica que el algoritmo no está evitando el peor desempeño posible. Además, está muy por debajo de las solución óptima, lo que señala que hay una diferencia significativa entre el rendimiento actual del modelo y el rendimiento potencial en los escenarios más favorables.

- **Proximal Policy Optimization (PPO)**: A pesar de las expectativas elevadas para este algoritmo, basadas en la literatura que sugiere su eficacia en entornos con alta incertidumbre, los resultados obtenidos no fueron consistentes. Se observaron patrones en los que la curva de recompensa experimentaba una tendencia ascendente, solo para descender abruptamente en etapas posteriores del entrenamiento. Esta oscilación podría atribuirse a varios factores:

  a. **Ajuste de Hiperparámetros**: La configuración inicial se centró principalmente en variar la cantidad de episodios y el parámetro de recorte (clipping). Es posible que un espacio más amplio de hiperparámetros necesite ser explorado para afinar el comportamiento del algoritmo en este entorno específico.

  b. **Adaptabilidad en Entornos Estocásticos**: Aunque PPO está diseñado para manejar incertidumbre, la naturaleza específica de este entorno, con su complejidad inherente y construcciones estocásticas, podría requerir adaptaciones adicionales en el algoritmo o en su configuración.

  c. **Convergencia Prematura**: La naturaleza del RL a veces puede llevar a la convergencia prematura a óptimos locales. La dinámica observada, donde la curva de recompensa ascendía inicialmente pero luego declinaba, podría ser indicativa de este fenómeno.

  Con base en estos hallazgos, es esencial que futuras investigaciones en PPO para este problema consideren una exploración más extensa y rigurosa de hiperparámetros, así como potencialmente adaptar o modificar aspectos del algoritmo para acomodar las peculiaridades del entorno en cuestión.

Modelo de 10 stands - 27 escenarios

En el contexto de un modelo con 10 stands y 27 escenarios, se ha evaluado el desempeño de **Double Dueling DQN junto con Prioritized Experience Replay**. Los resultados muestran un rendimiento significativamente bajo, que podría atribuirse a múltiples factores:

- La **complejidad del espacio de estado** se ve aumentada por el gran número de escenarios, dificultando la identificación y aprendizaje de la política óptima.

- La **variabilidad entre escenarios** introduce una mayor incertidumbre, requiriendo una estrategia de exploración y explotación más robusta.

- El **ajuste de hiperparámetros** puede necesitar una recalibración para acomodar el aumento en la complejidad del problema.

- La **capacidad del modelo** podría ser insuficiente, sugiriendo que podría ser necesario un incremento en la profundidad de la red neuronal o en la cantidad de neuronas por capa.
- Una **función de recompensa inadecuada** que no proporciona señales claras para guiar al algoritmo hacia el aprendizaje efectivo en un entorno con múltiples escenarios.
- La cantidad de **tiempo de entrenamiento** podría ser insuficiente para que el algoritmo converja hacia un rendimiento óptimo.
- La necesidad de **una mayor cantidad de experimentos** para validar la robustez y consistencia del algoritmo en el entorno propuesto.

Modelo de 16 stands

En esta instancia, se efectuaron entrenamientos focalizados en el algoritmo que demostró el mejor rendimiento en el modelo más compacto. Es necesario señalar que, para este modelo más extenso, el tiempo requerido para el entrenamiento se incrementó sustancialmente. Concretamente, se realizaron entrenamientos para el algoritmo:

- **Double Dueling DQN**: Se puede observa un comportamiento similar al caso de 10 stands. Donde existe una convergencia prematura a un nivel de recompesna que es del orden de los 4 millones, siendo aún así lejano al valor de la solución óptima. Además de presentar una alta variabilidad en la recompensa, siendo este ruido asociado al ya mencionado sampleo de los diferentes precios que construyen la trayectoria o escenario.
- **Double Dueling DQN con Prioritized Experience Replay**:
  1. La **expansión del espacio de estados** debido al aumento de stands ha incrementado la complejidad del modelo, lo cual plantea desafíos adicionales para la convergencia y la estabilidad del algoritmo de aprendizaje por refuerzo.
  2. La inclusión de **Prioritized Experience Replay (PER)** ha mejorado ligeramente el proceso de aprendizaje pero no ha logrado alcanzar la solución óptima. Esto sugiere la necesidad de una mayor optimización en la metodología de priorización y posiblemente una expansión en la capacidad del modelo para manejar la complejidad aumentada.
  3. La falta de suficientes **experimentos para validar la robustez** del algoritmo se hace evidente, lo que podría ayudar a comprender mejor el comportamiento del algoritmo y a identificar áreas clave para la mejora.
  4. Se necesita un **ajuste más riguroso de la arquitectura de la red**, la función de recompensa y la estrategia de balance entre exploración y explotación para abordar adecuadamente la complejidad introducida por el mayor número de stands.

Es imperativo subrayar que la naturaleza de las curvas de recompensa está intrínsecamente influenciada por la trayectoria muestreada. En otras palabras, el escenario construido durante la interacción con el ambiente dicta en gran medida las fluctuaciones observadas en las curvas. Esta variabilidad intrínseca subraya la importancia de adoptar un enfoque riguroso y sistemático al evaluar la eficacia de estos algoritmos en ambientes estocásticos.

Pese a las diferentes instancias de entrenamiento, ninguno de los algoritmos logró converger al óptimo esperado. Para entender este comportamiento, es esencial considerar varios factores subyacentes:

1. **Construcción del Ambiente**: La complejidad inherente de un ambiente estocástico con cotas de demanda, combinada con la incorporación de un árbol de escenarios de precios, puede haber introducido niveles de incertidumbre y volatilidad que complican el aprendizaje. Es plausible que la configuración del ambiente no facilitara la exploración suficiente de los estados, limitando así el descubrimiento de políticas más óptimas.

2. **Arquitectura de Red**: Si bien las redes neuronales son poderosas herramientas de aproximación, su diseño y arquitectura son cruciales para su desempeño en tareas específicas. Una arquitectura inadecuada o subóptima podría no capturar adecuadamente la estructura del problema, resultando en subóptimas estrategias de toma de decisiones.

3. **Función de Recompensa**: La forma y estructura de la función de recompensa juegan un papel vital en guiar el aprendizaje del agente. Si la recompensa es demasiado escasa o no proporciona señales claras, el agente podría encontrar dificultades para discernir acciones beneficiosas de las perjudiciales. Es fundamental asegurar que la recompensa refleje adecuadamente los objetivos del problema.

4. **Exploración frente a Explotación**: El balance entre explorar nuevas acciones y explotar acciones conocidas es un dilema clásico en el aprendizaje por refuerzo. Una estrategia de exploración insuficiente podría resultar en la convergencia a óptimos locales en lugar del óptimo global deseado.

5. **Variabilidad del Muestreo**: Dada la naturaleza estocástica del ambiente, el muestreo de escenarios tiene un papel crítico. Si los escenarios muestreados no son representativos de la diversidad real del problema, el agente podría aprender políticas que sean subóptimas en escenarios no observados.

6. **Hiperparámetros y Entrenamiento**: La elección de hiperparámetros y la duración del entrenamiento también influyen en la convergencia. Un entrenamiento insuficiente o hiperparámetros mal ajustados podrían impedir que el algoritmo alcance su potencial.

7.1.2. Ambiente 2: Demanda Irrestricta

El ambiente con demanda irrestricta se utiliza para estudiar el comportamiento de los algoritmos en un contexto sin las restricciones tradicionales de demanda. La falta de estas restricciones puede proporcionar información sobre la robustez de los algoritmos y su comportamiento en circunstancias más generales.

10 stands

- **Dueling DQN**: En el marco de la demanda irrestricta, la función de recompensa, al emplear Dueling DQN, tiende a aproximarse a valores que son cercanos al óptimo encontrado en problemas restrictos. No obstante, es imperativo recalcar que la comparación directa entre ambos puede ser engañosa, ya que, a pesar de sus semejanzas, son esencialmente problemas distintos. Este modelo irrestricto ofrece la posibilidad de discernir cómo se relacionan las recompensas con el muestreo de diferentes escenarios de precios. Aunque esto pueda parecer prometedor, el entrenamiento no muestra una tendencia clara hacia la convergencia. Se identifica, en cambio, una variabilidad pronunciada en la función de recompensa, que parece alinearse estrechamente con las cotas específicas de cada escenario evaluado. Cabe destacar que, a pesar de la flexibilidad que ofrece

un ambiente irrestricto, este escenario podría no ser completamente representativo o proveer inferencias profundas cuando se enfrenta a contextos operativos más realistas y definidos.

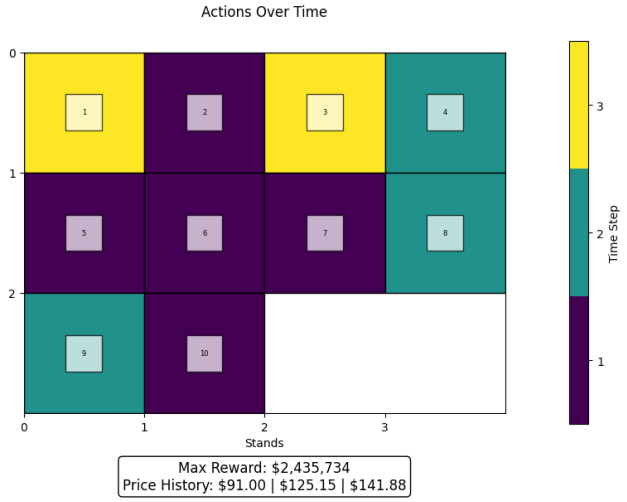
- **Proximal Policy Optimization (PPO)**: En contraste, utilizando PPO, se manifiesta una clara convergencia de las trayectorias de recompensa. Si bien solo se varió la cantidad de episodios, es destacable que al relajarse las restricciones de demanda, la convergencia a valores cercanos al óptimo no necesariamente proporciona claridad analítica. A pesar de ello, este comportamiento sugiere que PPO, en un ambiente irrestricto, posee potencial para una inicialización más eficiente. Por lo tanto, se podría inferir que, con una adecuada sintonización de parámetros en un ambiente restringido, PPO podría alcanzar los resultados deseados originalmente.

23 stands

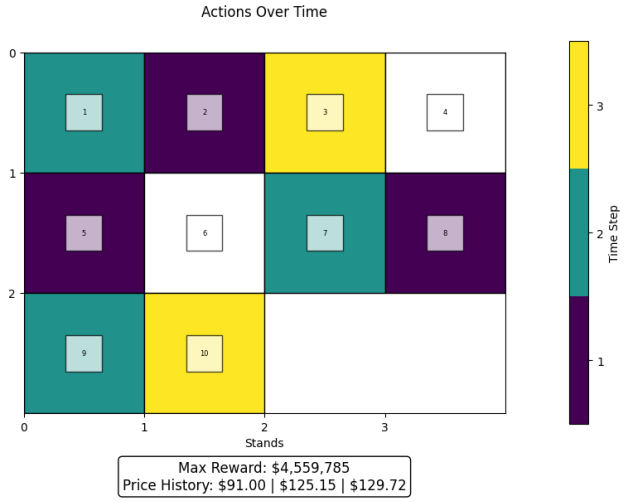
- **Proximal Policy Optimization**: Para esta configuración más ampliada de 23 stands, se percibe una tendencia ascendente en la curva de recompensa utilizando PPO. Sin embargo, se llevó a cabo sólo durante 5,000 episodios. Dada la limitada cantidad de episodios y la ausencia de una sintonización exhaustiva de hiperparámetros, es imperativo ser cauteloso antes de extraer conclusiones definitivas. Una exploración más extensa podría ser necesaria para obtener un panorama más completo de la eficacia del algoritmo en este contexto.

En resumen, mientras que el ambiente con demanda irrestricta ofrece insights sobre el comportamiento de los algoritmos en un escenario desregulado, es esencial interpretar los resultados con precaución y considerar las limitaciones inherentes de tal configuración en el contexto de la problemática real de cosecha de madera.

7.2. Discusión sobre la Convergencia de Políticas, acciones de máxima recompensa y desempeño



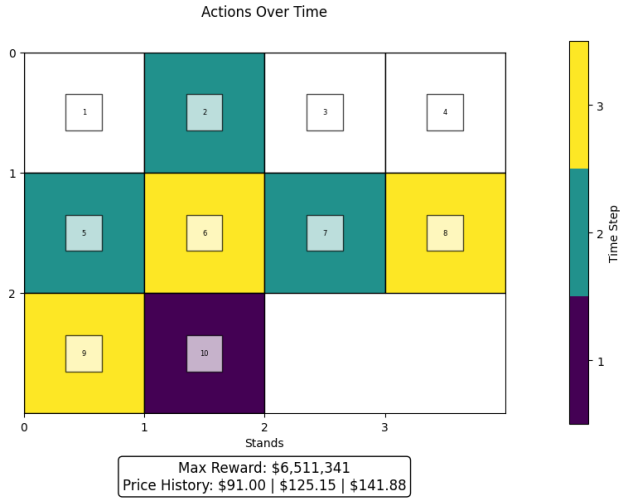
(a) Acción máxima recompensa entrenamiento DQN



(a) Acción máxima recompensa entrenamiento Dueling DQN -NN Arch 1

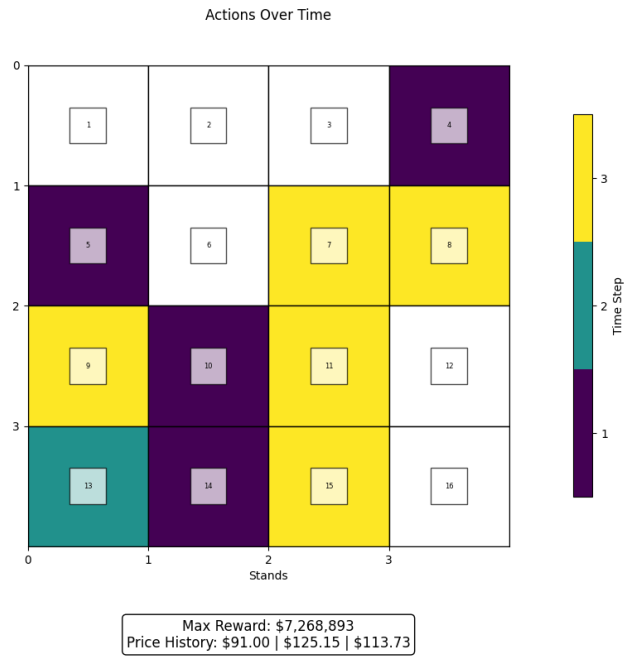
(a) Acción máxima recompensa entrenamiento Dueling DQN -NN Arch 2

(a) Acción máxima recompensa entrenamiento Dueling DQN -NN Arch 3



(a) Acción máxima recompensa entrenamiento Dueling Double DQN

(a) Acción máxima recompensa entrenamiento Dueling Double DQN w/ PER



(a) Acción máxima recompensa entrenamiento Dueling Double DQN 16 stands

Durante el proceso de entrenamiento de algoritmos, particularmente aquellos basados en Q-Learning como el DQN y sus variantes, es frecuente observar fluctuaciones en la calidad de las políticas aprendidas a medida que avanza el tiempo. A pesar de que en ciertos momentos se observó que las acciones máximas aproximaban o incluso superaban un desempeño óptimo, no se logró la convergencia hacia una política completamente óptima. Las razones potenciales detrás de este comportamiento son múltiples:

1. **Dilema Exploración vs. Explotación:** La tensión intrínseca en la dinámica de explorar nuevas estrategias frente a explotar estrategias conocidas puede resultar en la selección de acciones que no son óptimas durante el entrenamiento. Es crucial adecuar el factor de exploración a lo largo del proceso de aprendizaje.

2. **Inestabilidad en el Aprendizaje:** A pesar de los esfuerzos por estabilizar el proceso, como la memoria de repetición de experiencias y el uso de una Q-red objetivo, los algoritmos basados en Q-Learning pueden ser susceptibles a inestabilidades debido a correlaciones entre muestras y la no estacionariedad de los datos.

3. **Aproximación de la Función Q:** La arquitectura neuronal empleada para aproximar la función Q puede no ser la óptima para el problema en cuestión. Es esencial evaluar la capacidad y complejidad de la red en relación con la tarea.

4. **Tasa de Aprendizaje:** Una elección inadecuada de la tasa de aprendizaje puede decelerar la convergencia o incluso impedirla.

5. **Naturaleza de las Recompensas:** En entornos donde las recompensas son escasas o difusas, el agente puede enfrentar retos para asociar efectivamente sus acciones con las consecuentes recompensas.

6. **Variabilidad del Entorno:** Entornos con alta estocasticidad o variabilidad pueden presentar desafíos adicionales para la generalización de la política aprendida.

7. **Horizonte Temporal:** La elección del factor de descuento influirá en cómo el agente valora las recompensas a corto y largo plazo. Es imperativo ajustar este factor considerando el horizonte temporal deseado para la toma de decisiones.

7.3. Sobre el modelamiento de la incertidumbre y el ambiente

El enfoque adoptado en el modelado de este problema dejo en evidencia aspectos cruciales relacionados con la representación de la incertidumbre y la definición del espacio de estados y acciones. Es esencial resaltar las siguientes observaciones y reflexiones:

1. **Integración de Escenarios de Demanda y Precios**: El modelado del problema mediante escenarios de demanda pudo haber impuesto restricciones excesivas, incrementando el nivel de incertidumbre. La configuración adoptada para los escenarios de precios, al ser modelada independientemente de las demandas (las cuales se derivaron de los rendimientos de cada celda, teniendo en cuenta tasas de crecimiento y descuento), puede haber introducido desacoples en la representación del sistema. Desde una perspectiva de investigación operativa, es primordial garantizar que los escenarios de incertidumbre estén interrelacionados y se alimenten mutuamente para reflejar fielmente la dinámica del problema real.

2. **Modelado Determinístico**: Al realizar el entrenamiento de los algoritmos considerando cotas de demandas de manera determinista reduce en una dimensión el nivel de incertidumbre, este enfoque introdujó restricciones adicionales, limitando el espacio de acción y decisiones viables. En el contexto del Reinforcement Learning, un espacio de acción reducido puede limitar la exploración y, por ende, la capacidad del agente para aprender políticas óptimas. Dicho esto, se desistió de usar esta tercera variante del ambiente.

3. **Perspectivas para el Futuro en Modelado Estocástico**: Para futuras investigaciones, resulta necesario adoptar un enfoque de modelado más sofisticado. Esto implica la elaboración de un modelo que integre coherentemente demandas y precios, asegurando que ambos aspectos estén intrínsecamente ligados.

Capítulo 8

Conclusiones

8.1. Principales Resultados

La presente investigación se ha centrado en el aprendizaje por refuerzo aplicado a problemas de optimización, particularmente en el contexto cosecha forestal considerando restricciones de demanda y demanda irrestricta. A partir de esta exploración, se han identificado varios hallazgos esenciales:

1. En el análisis de redes DQN y sus variantes, se han identificado comportamientos variables en cuanto a su convergencia y estabilidad. El entrenamiento que utiliza el algoritmo Double Dueling DQN se ha distinguido por mostrar un rendimiento comparativamente superior, especialmente en escenarios con restricciones de demanda. A pesar de que la integración de Prioritized Experience Replay (PER) ha ofrecido ciertas mejoras en el aprendizaje, estas no han sido suficientes para lograr la solución óptima. Aunque sobresale dentro de su grupo de algoritmos comparativos, aún no satisface el objetivo de encontrar la política óptima que se desea alcanzar.

2. A pesar de la prometedora teoría detrás del algoritmo PPO, su implementación práctica careció de la consistencia esperada. Las fluctuaciones presentes en las curvas de recompensa subrayan la importancia de un ajuste de parámetros más minucioso.

3. Se reafirma que el intrincado diseño del ambiente, dictado por su comportamiento estocástico y la integración de un árbol de escenarios de precios, plantea ciertos desafíos al empleo del RL en la optimización.

8.2. Implicaciones Teóricas

La investigación resalta la crítica necesidad de la adaptabilidad y ajuste preciso de los algoritmos de RL en entornos de alta incertidumbre. Cada entorno posee retos singulares, descartando la existencia de un algoritmo universal que asegure la óptima actuación en cualquier circunstancia.

La inherente variabilidad, que se refleja en las curvas de recompensa y es influenciada por el muestreo estocástico, enfatiza el requerimiento de estrategias robustas en la evaluación de algoritmos de RL. Basarse exclusivamente en criterios como la recompensa acumulada podría obviar la multidimensionalidad inherente a la dinámica de aprendizaje.

1. **Dinámica de los Algoritmos de RL en Entornos Estocásticos**: La variabilidad y la incertidumbre inherentes en entornos estocásticos, como los presentes en la cosecha forestal, exigen un enfoque de RL que no solo se centre en la maximización de la recompensa a corto plazo, sino también en la adaptabilidad y robustez a largo plazo. Esto implica un énfasis renovado en el ajuste de hiperparámetros y en el diseño de arquitecturas de red que puedan manejar fluctuaciones en los datos de entrada y en las recompensas obtenidas.

2. **Balance entre Exploración y Explotación**: En problemas de optimización estocástica, el balance adecuado entre exploración y explotación se convierte en un factor crítico. La capacidad de un algoritmo para explorar eficazmente el espacio de estados y acciones bajo incertidumbre, sin quedarse atrapado en óptimos locales o subóptimos, es crucial para el descubrimiento de estrategias de decisión óptimas.

3. **Modelado y Aproximación de Funciones de Valor**: La precisión en la estimación de las funciones de valor en RL es un aspecto fundamental en entornos estocásticos. El uso de técnicas avanzadas para la aproximación de funciones, como las redes neuronales profundas, debe ser cuidadosamente calibrado para evitar el sobreajuste y garantizar la generalización en diferentes escenarios de incertidumbre.

4. **Importancia de la Simulación y el Análisis de Sensibilidad**: Dado el comportamiento dinámico y a menudo impredecible de los sistemas estocásticos, la simulación juega un papel crucial en el desarrollo y la evaluación de algoritmos de RL. El análisis de sensibilidad, que examina cómo las variaciones en los parámetros del sistema afectan los resultados del aprendizaje, es esencial para comprender y mejorar la robustez de los algoritmos de RL.

8.3. Trabajo Futuro

Dados los resultados obtenidos, es evidente que existen diversas áreas de mejora para el modelo y la implementación actual. Las siguientes propuestas podrían ser consideradas en investigaciones y desarrollos futuros:

1. **Mejorar la Función de Recompensa**: La función de recompensa es esencial en cualquier problema de RL. Una función más adaptativa y robusta, que considere la estocasticidad y particularidades del problema de cosecha de madera, podría mejorar significativamente la calidad de las políticas aprendidas.

    - **Robustez frente a la Incertidumbre**: Una función de recompensa bien definida debe permitir que el agente tome decisiones que sean robustas frente a las incertidumbres inherentes al problema. Esto significa que, en lugar de optimizar para el mejor escenario posible o el peor escenario posible, el agente debería ser capaz de tomar decisiones que sean razonablemente buenas en una amplia gama de escenarios posibles.
    - **Importancia de la Exploración en Entornos Estocásticos**: En entornos con incertidumbre, la exploración se vuelve aún más crucial. La función de recompensa debe estar diseñada para incentivar al agente a explorar diferentes acciones y estrategias, especialmente en las etapas iniciales de aprendizaje. Esto permitirá al agente

obtener una comprensión más completa del entorno y las recompensas asociadas a diferentes acciones en diferentes escenarios.

- **Feedback Continuo y Adaptabilidad**: Dada la naturaleza cambiante de los precios y la demanda en este problema, es vital que la función de recompensa proporcione un feedback continuo y permita al agente adaptarse a los cambios en el entorno. Una función de recompensa estática podría no ser suficiente; en su lugar, una función dinámica que se adapte con el tiempo podría ser más apropiada.

2. **Búsqueda Mejorada de Hiperparámetros**: Es fundamental realizar una búsqueda más exhaustiva y refinada de hiperparámetros. Una técnica a considerar es el *fine-tuning*, que implica ajustar ligeramente los hiperparámetros de un modelo preentrenado en lugar de entrenar un modelo desde cero. Esta técnica aprovecha el conocimiento previo adquirido por el modelo, permitiendo lograr mejoras con un menor costo computacional.

3. **Ampliación en la Optimización con PPO**: Aunque se utilizó el algoritmo Proximal Policy Optimization (PPO) en la investigación, las limitaciones computacionales restringieron la diversidad en la configuración de hiperparámetros. Una exploración más detallada podría revelar políticas que converjan eficientemente hacia soluciones óptimas.

4. **Incorporación de DDQN from Demonstrations**:

   El algoritmo DDQN from Demonstrations [20] es una extensión del tradicional Deep Q-Network (DQN) que integra aprendizaje de demostraciones. Esta técnica se basa en la idea de mejorar el proceso de aprendizaje mediante la utilización de un conjunto de políticas óptimas o subóptimas demostradas previamente. Las ventajas técnicas de emplear este enfoque en el contexto del problema de cosecha de madera son múltiples:

   - **Aceleración del Aprendizaje**: La integración de demostraciones en el proceso de entrenamiento puede acelerar significativamente el aprendizaje, especialmente en las etapas iniciales. Al proporcionar ejemplos de políticas efectivas, el agente puede rápidamente aprender estrategias útiles, reduciendo el tiempo necesario para explorar el espacio de acciones de manera aleatoria.
   - **Mejora en la Exploración**: Al contar con demostraciones de políticas exitosas, el agente tiene la oportunidad de explorar partes del espacio de estado y acción que de otro modo podrían ser ignoradas o infravaloradas. Esto es particularmente valioso en entornos estocásticos y complejos, donde la identificación de acciones óptimas puede no ser intuitiva.
   - **Reducción de la Varianza en la Aprendizaje**: La utilización de demostraciones puede ayudar a estabilizar el proceso de aprendizaje reduciendo la varianza inherente al método de prueba y error. Al tener acceso a comportamientos demostrados, el agente puede evitar la toma de decisiones extremadamente pobres o erráticas.
   - **Superación de Óptimos Locales**: En problemas complejos como la cosecha de madera, donde existen múltiples óptimos locales, el uso de demostraciones puede guiar al agente a superar estos óptimos y explorar soluciones potencialmente más efectivas.
   - **Facilitación del Ajuste de Hiperparámetros**: La incorporación de demostraciones puede también facilitar el proceso de ajuste de hiperparámetros, ya que proporciona un conjunto de datos inicial más informativo y orientado hacia políticas eficaces.

5. **Explorar Nuevas Estructuras de Redes Neuronales**

   La estructura de la red neuronal juega un papel crucial en la eficacia del aprendizaje de políticas. Dado que el problema en cuestión presenta una naturaleza secuencial, las Redes Neuronales Recurrentes (RNN) emergen como una elección teóricamente fundada. Las RNN tienen la capacidad de retener información de estados anteriores en secuencias temporales, lo que las hace especialmente adecuadas para modelar problemas donde el contexto histórico, como los precios y demandas pasadas en este caso, influye en las decisiones futuras. Esta memoria intrínseca permite que las RNN capturen dinámicas temporales y relaciones a largo plazo, lo cual es esencial para el problema de cosecha de madera. Aunque existen otras arquitecturas, como las Redes Neuronales Convolucionales (CNN), las RNN se presentan como la opción primordial para este escenario, garantizando una representación más coherente y precisa de las secuencias temporales en cuestión. [21]

6. **Experimentación con Horizontes de Tiempo más Extensos**: Extender el horizonte de tiempo podría proporcionar una perspectiva más completa de las implicaciones a largo plazo de las políticas aprendidas. Aunque esto puede aumentar la complejidad y el tiempo de entrenamiento, los beneficios en términos de políticas más robustas y sostenibles podrían ser significativos.

7. **Implementación de Neural Progressive Hedging (NPH)**: El Neural Progressive Hedging es una técnica que combina la teoría de la optimización estocástica con el aprendizaje profundo [22]. Al incorporar redes neuronales en el algoritmo de PH, es posible modelar y resolver problemas estocásticos complejos con una gran cantidad de escenarios. Esta técnica podría ser particularmente útil para el problema de cosecha de madera, dada la estocasticidad y las múltiples fuentes de incertidumbre involucradas. Implementar y experimentar con NPH podría ofrecer soluciones más robustas y eficientes, aprovechando la capacidad del aprendizaje profundo para capturar relaciones complejas y no lineales en el problema.

8. **Inclusión de Decisiones Adicionales al Modelo**: Se propone expandir el modelamiento del problema original, integrando la generación de caminos en conjunto con los escenarios de precios. Si bien esta expansión incrementará la complejidad del problema, proporcionará una representación más realista de su dinámica.

La implementación y evaluación de estas mejoras podría llevar el modelo a un nuevo nivel de precisión y eficiencia, permitiendo abordar el problema de cosecha de madera con una mayor profundidad y rigor.

8.4. Reflexiones Finales

El estudio ha arrojado luz sobre los retos y oportunidades asociados con la aplicación de técnicas de aprendizaje por refuerzo en problemas de optimización complejos. Aunque se identificaron áreas de mejora, los resultados también indican la viabilidad y el potencial de RL en este dominio. Es crucial continuar este camino de investigación, pues la promesa de soluciones automatizadas y optimizadas puede tener implicaciones significativas en múltiples campos, desde la gestión de recursos hasta la toma de decisiones estratégicas en entornos inciertos.

Bibliografía

[1] Bellman, R., Dynamic Programming. Princeton University Press, 1957.

[2] Alonso-Ayuso, A., Escudero, L., Guignard, M., *et al.*, "Forestry management under uncertainty," Annals of Operations Research, vol. 190, pp. 17–39, 2011, doi:10.1007/s10479-009-0561-0.

[3] Sutton, R. S. y Barto, A. G., Reinforcement learning: An introduction. MIT press, 2018.

[4] Bellman, R., "A markovian decision process," Journal of Mathematics and Mechanics, vol. 6, no. 5, pp. 679–684, 1957, http://www.jstor.org/stable/24900506.

[5] Puterman, M. L., Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, 1994.

[6] Watkins, C. J. C. H., Learning from Delayed Rewards. PhD thesis, University of Cambridge, England, 1989.

[7] Powell, W., Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions. Wiley, 2022, https://books.google.cl/books?id=6ahsEAAAQBAJ.

[8] Sutton, R. S. y Barto, A. G., Reinforcement Learning: An Introduction. The MIT Press, 2nd ed., 2018. Chapter 13: Policy Gradient Methods.

[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., y Klimov, O., "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[10] Sutton, R. S. y Barto, A. G., Reinforcement Learning: An Introduction, cap. Actor-Critic Methods, pp. 331–332. MIT Press, 2nd ed., 2018.

[11] van Hasselt, H., Guez, A., y Silver, D., "Deep reinforcement learning with double q-learning," en Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI, 2015.

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., y Riedmiller, M. A., "Playing atari with deep reinforcement learning," ArXiv, vol. abs/1312.5602, 2013, https://api.semanticscholar.org/CorpusID:15238391.

[13] Rockafellar, R. y Wets, R. J.-B., "Scenario and policy aggregation in optimisation under uncertainty," Mathematics of Operations Research, vol. 16, pp. 119–147, 1991.

[14] Knueven, B., Mildebrath, D., Muir, C., Siirola, J. D., Watson, J.-P., y Woodruff, D. L., "A parallel hub-and-spoke system for large-scale scenario-based optimization under uncertainty," 2020.

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves,

A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., y et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[16] Wang, Z., Freitas, N., y Lanctot, M., "Dueling network architectures for deep reinforcement learning," 2015.

[17] Hasselt, H. V., Guez, A., y Silver, D., "Deep reinforcement learning with double q-learning," en AAAI Conference on Artificial Intelligence, 2015, https://api.semanticscholar.org/CorpusID:6208256.

[18] et al., S. G., "TF-Agents: A library for reinforcement learning in tensorflow," 2020, https://github.com/tensorflow/agents. Accedido: 2023-11-13.

[19] Schulman, J., Levine, S., Abbeel, P., Jordan, M., y Moritz, P., "Trust region policy optimization," International Conference on Machine Learning, pp. 1889–1897, 2015.

[20] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J. Z., Gruslys, A., Molloy, W., Czarnecki, W. M., Silver, D., y Lever, G., "Deep q-learning from demonstrations," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.

[21] Goodfellow, I., Bengio, Y., y Courville, A., Deep Learning. Cambridge, MA: MIT Press, 2016.

[22] Ghosh, S., Wynter, L., Lim, S. H., y Nguyen, D. T., "Neural-progressive hedging: Enforcing constraints in reinforcement learning with stochastic programming," 2022.

Anexo

Generación de parámetros del bosque

Año	**Madera Aserrada** [$\$/m^3$]
1987	40
1988	49
1989	51
1990	55
1991	66
1992	91
1993	91
1994	91
1995	99
1996	102
1997	100
1998	90
1999	80
2000	76
2001	61
2002	57
2003	58
2004	67
2005	91
2006	100
2007	103
2008	105
2009	97
2010	114
2011	119
2012	119
2013	118
2014	106
2015	95

2016	93
2017	101
2018	102
2019	93
2020	89
2021	151
2022	155

Tabla .1: Valores de Madera Aserrada para cada año. (Fuente: INSTITUTO FORESTAL: Precios forestales. Boletín N°181 Junio 2022. Tabla: PRECIOS NOMINALES TROZAS Y MADERA DE PINO RADIATA MERCADO INTERNO 1987 - 2022

Para cada una de las implementación se utilizo este árbol de escenarios de precios. Para crear este, se considera como precio inicial 91 [$\$/m^3$] y una $\sigma = 0.25$. Estos valor corresponde a al promedio de la madera aserrada entre 1987 y 2022.

Escenario	$P_{t=1}$ **[$/m^3]**	$P_{t=2}$ **[$/m^3]**	$P_{t=3}$ **[$/m^3]**
1	91.00	125.15	113.73
2	91.00	125.15	129.72
3	91.00	125.15	141.88
4	91.00	99.87	94.12
5	91.00	99.87	96.32
6	91.00	99.87	99.84
7	91.00	80.99	89.44
8	91.00	80.99	91.34
9	91.00	80.99	68.44

Tabla .2: Valores de precios para cada período de tiempo.

Escenario	$P_{t=1}$ **[$m^3]**	$P_{t=2}$ **[$m^3]**	$P_{t=3}$ **[$m^3]**	$P_{t=4}$ **[$m^3]**
1	91.0	125.15	113.73	94.63
2	91.0	125.15	113.73	106.97
3	91.0	125.15	113.73	128.3
4	91.0	125.15	129.72	135.55
5	91.0	125.15	129.72	107.15
6	91.0	125.15	129.72	125.92
7	91.0	125.15	141.88	153.6
8	91.0	125.15	141.88	130.27

9	91.0	125.15	141.88	116.95
10	91.0	99.87	94.12	117.31
11	91.0	99.87	94.12	102.67
12	91.0	99.87	94.12	92.79
13	91.0	99.87	96.32	114.68
14	91.0	99.87	96.32	86.19
15	91.0	99.87	96.32	75.39
16	91.0	99.87	99.84	87.82
17	91.0	99.87	99.84	154.85
18	91.0	99.87	99.84	142.82
19	91.0	80.99	89.44	88.16
20	91.0	80.99	89.44	121.92
21	91.0	80.99	89.44	81.38
22	91.0	80.99	91.34	67.81
23	91.0	80.99	91.34	82.34
24	91.0	80.99	91.34	95.69
25	91.0	80.99	68.44	65.41
26	91.0	80.99	68.44	74.04
27	91.0	80.99	68.44	89.66

Tabla .3: Valores de precios para cada período de tiempo (4 años).

Para el caso de los otros parámetros del bosque se considera

Parámetro	**Valor**
Número de stands	10
Horizonte de Planificación	3
Tasa de crecimiento	5 %
Tasa de descuento	5 %
Demanda mínima	40 %
Demanda máxima	80 %
Área por stand	25 [Ha]
Productividad promedio por hectárea	400 [m^3]
Costo de cosecha por m^3	\$40
Costo por hectárea promedio	\$100
Costo fijo promedio	\$2000

Tabla .4: Valores de los parámetros para el bosque.

El proceso de cálculo de costos y productividades comienza con la generación de valores iniciales para el primer período de tiempo ($t = 0$) de manera estocástica en virtud de darle variación a cada stand. Para cada uno de estos, la productividad inicial se obtiene aleatoriamente de una distribución normal con una media igual a la productividad promedio por hectárea. La desviación estándar de esta distribución se toma como una décima parte de la media, lo que introduce variabilidad en las productividades entre los stands. Luego, multiplicando la productividad por el área de cada stand, se calcula la cantidad total de madera producida en el primer período.

Del mismo modo, los costos iniciales por stand se generan aleatoriamente utilizando distribuciones normales con medias correspondientes a los costos por hectárea y costos fijos. La desviación estándar para ambas distribuciones también es un décimo de la media, lo que da heterogeneidad al experimento.

A medida que avanzan los períodos de tiempo, los valores de productividad y costos varían en cada stand, ya que se aplican tasas de crecimiento y descuento previamente definidas para actualizarlos. Este proceso permite simular la evolución dinámica de estos parámetros a lo largo del tiempo, reflejando así la gestión y dinámicas de crecimiento.

Tabla .5: Productividad [m^3] para cada stand en diferentes períodos de tiempo.

Stand	**Período 1**	**Período 2**	**Período 3**
1	10472	10995	11544
2	8299	8713	9148
3	10005	10505	11030
4	12871	13514	14189
5	11095	11649	12231
6	9877	10370	10888
7	8970	9418	9888
8	10138	10644	11176
9	8845	9287	9751
10	8211	8621	9052

Tabla .6: Costos $/m^3 para cada stand en diferentes períodos de tiempo.

Stand	**Período 1**	**Período 2**	**Período 3**
1	14850	14107	13401
2	12680	12046	11443
3	14313	13597	12917
4	17316	16450	15627
5	15559	14781	14041
6	14447	13724	13037
7	13872	13178	12519
8	14903	14157	13449
9	12455	11832	11240
10	12026	11424	10852

La demanda mínima y máxima por período de tiempo se calculan a través de un procedimiento iterativo. Al principio, para el primer período de tiempo (t=0), estas demandas se determinan multiplicando la demanda mínima y máxima respectivas por la suma de todas las producciones del primer período, dividiendo luego por el número total de períodos de tiempo. Esto genera un valor de demanda proporcional al total de producción para ese período y repartido de manera equitativa entre todos los períodos.

En los períodos siguientes, la demanda mínima y máxima para cada período de tiempo t se calculan como el valor de la demanda mínima y máxima del período anterior (t-1), escalado por un factor de crecimiento (la tasa de crecimiento). De esta forma, se considera el crecimiento en la demanda de un período al siguiente.

Este procedimiento asegura que los valores de demanda están en correspondencia con los niveles de producción y cambian de acuerdo con un factor de crecimiento determinado, reflejando la dinámica temporal en la demanda.

Tabla .7: Demanda mínima y máxima en diferentes períodos de tiempo.

	Período 1 [m^3]	**Período 2 [m^3]**	**Período 3 [m^3]**
D_{min}	19756	20743	21780
D_{max}	26342	27659	29041

Estas cotas son utilizadas de dos maneras:

- Se calculan para estos valores la media y desviación estándar para generar un árbol de escenarios basado en las demandas según nivel de producción.
- Se utilizan en uno de los experimentos en el cual se define utilizan directamente como cotas de demandas para el agente.

Escenario	**Periodo 1**		**Periodo 2**		**Periodo 3**	
	D_{min}	D_{max}	D_{min}	D_{max}	D_{min}	D_{max}
1	22986	24466	21846	23459	20831	21790

Tabla .8 – *Continuación de la página anterior*

Escenario	**Periodo 1**		**Periodo 2**		**Periodo 3**	
	D_{min}	D_{max}	D_{min}	D_{max}	D_{min}	D_{max}
2	22986	24466	21846	23459	20831	21790
⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	22986	24466	23486	24679	21515	22326

Tabla .8: Límites de demanda para diferentes escenarios y períodos de tiempo.