ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

APLICACIÓN DE LAS TÉCNICAS DE OPTIMIZACIÓN ESTADÍSTICA ENFOCADA A LA COMPUTACIÓN GRÁFICA

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE MATEMÁTICO APLICADO

GEOCONDA DENNISSE MOLINA MORALES

geoconda.molina@epn.edu.ec

DIRECTOR: MÉNTHOR OSWALDO URVINA MAYORGA

menthor.urvina@epn.edu.ec

5 DE ABRIL DE 2024

CERTIFICACIONES

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

_____

Geoconda Dennisse Molina Morales

Certifico que el presente trabajo de integración curricular fue desarrollado por Geoconda Dennisse Molina Morales, bajo mi supervisión.

_____

Ménthor Oswaldo Urvina Mayorga

DIRECTOR

DECLARACIÓN DE AUTORÍA

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

Geoconda Dennisse Molina Morales

Ménthor Oswaldo Urvina Mayorga

RESUMEN

(Máximo 250 palabras)

Palabras clave: palabra1, palabra2, palabra3,. . ., palabra6.

ABSTRACT

(Máximo 250 palabras)

Keywords: palabra1, palabra2, palabra3,. . ., palabra6.

Índice general

Índice de figuras

Capítulo 1

Descripción del componente desarrollado

1.1. Descripción del Proyecto

Para explicar de mejor manera las aplicaciones de las técnicas de optimización estadística, los modelos estadísticos y los conceptos estadísticos en la computación gráfica, se plantea un proyecto desarrollado con el lenguaje de programación Python, el cual consiste en un agente entrenado por q- leearning del cual se derivan las explicaciones de estadística, y de una red neuronal convolucional (CNN), la cual es el modelo estadístico que procesa imágenes y que utiliza técnicas de optimización estadística para aproximar de mejor manera sus resultados, así, el agente q-learner conectado a la red CNN utiliza las salidas del modelo CNN para ajustar su función de calidad y así tomar elecciones que le permita tener un aprendizaje con cada decisión tomada en el entorno en el que se encuentra.

1.2. Objetivo general

Investigar y desarrollar un agente Q-learning que incorpore redes neuronales convolucionales para la realización de tareas en entornos gráficos, con el fin de demostrar cómo los matemáticos pueden contribuir al mundo del arte gráfico y la computación gráfica. Además, se resaltará el papel crucial de la optimización estadística proporcionada por los mate-

máticos en el desarrollo de inteligencias artificiales, especialmente en el campo del deep learning, con el objetivo de demostrar su relevancia en la mejora de la eficiencia y precisión en la resolución de problemas.

1.3. Objetivos específicos

- Desarrollar un agente q-learning que integre redes neuronales convolucionales.
- Crear un entorno virtual que permita al agente realizar diversas tareas de las cuales aprende sucesivamente.
- Recopilar datos mediante la observación y registro de las acciones y decisiones tomadas por el agente durante su proceso de aprendizaje en el entorno gráfico.
- Analizar cómo la integración de redes neuronales convolucionales y técnicas de optimización estadística influye en la capacidad de aprendizaje y ejecución de tareas del agente en entornos gráficos, identificando las mejoras significativas en términos de eficiencia y precisión.
- Promover la colaboración entre matemáticos y expertos en el desarrollo de tecnologías avanzadas.

1.4. Alcance

Este estudio se enfocará en el desarrollo e implementación de un agente de aprendizaje basado en el algoritmo Q-Learning, integrado con redes neuronales convolucionales, para la realización de tareas en entornos gráficos. El alcance del estudio abarcará la creación de un entorno virtual que simule situaciones y tareas, permitiendo la observación y registro de las acciones y decisiones tomadas por el agente durante su proceso de aprendizaje.

El período de estudio se extenderá desde la implementación del agente hasta la conclusión del análisis de los datos recopilados, con una duración estimada de cuatro meses.

Los datos utilizados en este estudio serán generados internamente por el entorno virtual y consistirán en registros de las interacciones del agente con su entorno gráfico. No se utilizarán datos de usuarios reales ni se recopilará información personal para este propósito.

Los métodos de análisis incluirán técnicas de optimización estadística para mejorar la eficiencia y precisión del agente en la toma de decisiones, así como la evaluación de su desempeño en la realización de actividades dentro del entorno gráfico. Además, se analizará cómo la integración de redes neuronales convolucionales y técnicas de optimización estadística influye en la capacidad de aprendizaje y ejecución de tareas del agente, identificando mejoras significativas en términos de eficiencia y precisión.

Capítulo 2

Marco teórico

2.1. Conceptos Fundamentales

2.1.1. Agente

Un agente es una entidad que toma decisiones y realiza acciones en un entorno con el objetivo de alcanzar ciertos objetivos o maximizar una recompensa acumulada a lo largo del tiempo. El agente puede ser cualquier sistema o programa que interactúe con su entorno, como un robot, un software de juego, etc.

2.1.2. Entorno

El entorno representa el mundo en el que el agente interactúa y toma decisiones. Puede ser cualquier sistema o situación en la que el agente esté operando, como un videojuego, un robot físico, o un entorno simulado. El entorno proporciona retroalimentación al agente en forma de recompensas o penalizaciones según las acciones que tome.

2.1.3. Estado

El estado es una representación del entorno en un momento dado. Proporciona información sobre la situación actual del agente, incluida

su ubicación, las condiciones del entorno y cualquier otra información relevante para la toma de decisiones.

Espacios de Estados Continuos:

Un tipo común de espacio de estados es el espacio continuo, representado matemáticamente como $\mathbb{R}^n$, donde n es la dimensión del espacio. En este tipo de espacio, cada componente x_i de un estado puede tomar cualquier valor real dentro de un rango específico. Por ejemplo, un espacio de estado Box puede definir un conjunto de puntos en un espacio bidimensional con coordenadas (x, y), donde cada coordenada puede variar de manera continua dentro de un rango predefinido.

Espacios de Estados Discretos:

Por otro lado, los espacios de estados discretos representan conjuntos finitos de valores, donde cada estado se identifica mediante un número entero dentro de un rango específico. Estos espacios se modelan matemáticamente como conjuntos discretos $\{0, 1, 2, ..., n-1\}$, donde n es el número total de elementos en el conjunto. Por ejemplo, un espacio de estado Discrete puede representar un conjunto de acciones posibles que un agente puede tomar, donde cada acción está asociada con un número entero dentro de un rango predefinido.

Espacios Compuestos:

Además de los espacios simples, existen espacios de estados compuestos que combinan múltiples sub-espacios de diferentes tipos. Por ejemplo, el espacio de tipo Dictionary permite representar observaciones que consisten en varios sub-espacios, cada uno de los cuales puede ser de un tipo diferente (Box, Discrete, etc.). Esto es útil para modelar observaciones complejas que requieren diferentes tipos de información. Por otro lado, el espacio de tipo Tuple también permite combinar múltiples subespacios, pero se diferencia en que preserva el orden de los elementos, similar a una tupla en Python. Estos espacios compuestos proporcionan una flexibilidad adicional para representar observaciones complejas en

problemas de RL.

Espacios de Estados Binarios:

El espacio de estados MultiBinary representa un vector binario de longitud fija, donde cada elemento puede ser 0 o 1. Matemáticamente, este espacio se representa como $\{0, 1\}^n$, donde n es la longitud del vector binario. Por ejemplo, un espacio de estado MultiBinary con longitud 3 puede representar todas las combinaciones posibles de 3 bits.

Espacios de Estados Multibinarios:

El espacio de estados MultiDiscrete representa un espacio de observación compuesto por varias dimensiones discretas, donde cada dimensión puede tener un rango diferente. A diferencia del espacio Discrete, donde todas las dimensiones comparten el mismo rango, en MultiDiscrete cada dimensión puede tener su propio rango de valores. Esto permite modelar observaciones multidimensionales con diferentes escalas o rangos de valores.

2.1.4. Acción

Una acción es cualquier movimiento o decisión que el agente puede realizar en respuesta a un estado dado. Las acciones pueden ser discretas o continuas, dependiendo de la naturaleza del problema y de las capacidades del agente.

2.1.5. Recompensa

La recompensa es una señal de retroalimentación que el entorno proporciona al agente en respuesta a una acción específica. La recompensa indica cuán beneficiosa o perjudicial fue la acción tomada por el agente en un estado dado y sirve como guía para el aprendizaje.

2.1.6. Política

La política de un agente es una estrategia que determina qué acción debe tomar el agente en un determinado estado del entorno. Es esencialmente una función que asigna acciones a estados con el fin de maximizar la recompensa esperada a largo plazo. La política puede ser determinista o estocástica, dependiendo de si las acciones son completamente predecibles o tienen cierta aleatoriedad.

2.2. Rainforcement Learning

También conocido como aprendizaje por refuerzo, constituye una rama fundamental dentro del ámbito del aprendizaje automático y el control óptimo. Este enfoque se centra en la toma de decisiones de agentes inteligentes en entornos dinámicos con el objetivo de maximizar la recompensa acumulativa. Junto con el aprendizaje supervisado (Machine Learning) y el aprendizaje no supervisado (Unsupervised Learning), el aprendizaje por refuerzo conforma uno de los paradigmas básicos del aprendizaje automático.

"A diferencia de los métodos de gradiente de políticas, que intentan aprender funciones que asignan directamente una observación a una acción, Q-Learning intenta aprender el valor de estar en un estado determinado y realizar una acción específica allí."(4)

En su esencia, el aprendizaje por refuerzo se preocupa por encontrar un equilibrio entre la exploración de territorios inexplorados y la explotación del conocimiento existente, todo ello con el fin de maximizar la recompensa a largo plazo. Es crucial destacar que la retroalimentación en este proceso puede ser incompleta o experimentar ciertos retrasos.

El entorno, en este contexto, se modela mediante un proceso de decisión de Markov (MDP). Muchos algoritmos de aprendizaje por refuerzo emplean técnicas de programación dinámica para abordar este tipo de

entorno.

La diferencia fundamental entre los métodos clásicos de programación dinámica y los algoritmos de aprendizaje por refuerzo se encuentra en su aproximación al conocimiento del modelo matemático exacto del proceso de decisión de Markov.

En la programación dinámica clásica, se parte del supuesto de tener un conocimiento preciso y completo del modelo matemático del proceso de decisión de Markov.(6) Esto implica conocer de antemano todas las transiciones posibles entre estados, así como las probabilidades asociadas y las recompensas esperadas. Estos métodos clásicos utilizan esta información para calcular de manera óptima las políticas de decisión, buscando la solución más eficiente para el problema.

En cambio, los algoritmos de aprendizaje por refuerzo no asumen previamente este conocimiento detallado del modelo. En lugar de eso, estos algoritmos están diseñados para aprender y adaptarse directamente del entorno mediante la interacción continua. Utilizan la experiencia acumulada a lo largo del tiempo para mejorar gradualmente su comprensión del modelo del proceso de decisión de Markov, ajustando sus estrategias de decisión para maximizar la recompensa a largo plazo. Este enfoque es particularmente valioso en situaciones donde obtener información completa y precisa sobre el modelo resulta difícil o impracticable, permitiendo así que los algoritmos de aprendizaje por refuerzo se desempeñen eficazmente en entornos dinámicos y complejos.

2.2.1. Ecuacion de Bellman

La ecuación de Bellman en el contexto del aprendizaje por refuerzo es una herramienta conceptual clave que nos ayuda a entender cómo asignar valores a acciones o estados en un entorno dinámico. En términos simples, esta ecuación establece que el valor de una elección o situación se compone de dos partes: la recompensa inmediata y la estimación del valor futuro.

Cuando tomamos una acción o nos encontramos en un estado específico, recibimos una recompensa instantánea. Sin embargo, la ecuación de Bellman reconoce que la toma de decisiones no solo se trata de recompensas inmediatas, sino también de cómo esas decisiones afectan las recompensas a largo plazo. Por lo tanto, el valor de una acción o estado incluye no solo la recompensa inmediata, sino también la expectativa de las recompensas futuras.

Esta expectativa de recompensas futuras se pondera mediante un factor de descuento. Este factor refleja la idea de que las recompensas futuras son menos valiosas que las recompensas inmediatas. Es decir, valoramos más una recompensa hoy que una recompensa de igual cantidad en el futuro.

En este contexto podemos observar la ecuación de Bellman de manera matemática de la siguiente forma(2):

Sean:

$$s = estados$$

$$a = acciones$$

$$R = recompensa$$

$$\gamma = factor\ descuento$$

Se considera los conjuntos $A = \{a_1, a_2, ..., a_n\}$ y $S = \{s_1, s_2, ..., s_n\}$ de acciones y estados, ambos finitos.

Inicialmente se tratará el caso para procesos deterministas, es decir, conocemos el valor $V(s')$ de ante mano para todos los estados futuros, así la ecuación se ve de la sigueinte manera:

$$V(s) = \max_a [R(s,a) + \gamma V(s')]$$

Por lo que, se calcula $V(s)$ tomando el máximo de las recompensas

obtenidas eligiendo la accion a, en el estado s más el factor de descuento γ por el valor futuro en el estado s' al cual se llega despues de tomar la acción a.

2.2.2. Ecuación de Bellman para procesos estocásticos

Debido a que en la vida real no existen procesos deterministas, se debe incluir la parte estocástica en la ecuación tradicional de Bellman. En un entorno estocástico, las acciones no conducen siempre al mismo resultado; en cambio, hay una probabilidad asociada con cada transición entre estados.

En este contexto, la ecuación de Bellman refleja la idea de que el valor de una elección o estado se calcula tomando en cuenta la recompensa inmediata y la expectativa de recompensas futuras, pero ahora incorpora la aleatoriedad de las transiciones entre estados. Esto significa que, al tomar una acción, no podemos predecir con certeza el próximo estado; en su lugar, debemos considerar las probabilidades asociadas con cada posible transición.(2)

El factor de descuento sigue siendo relevante en un entorno estocástico y sirve para ponderar las recompensas futuras, reconociendo que la incertidumbre puede afectar la magnitud de esas recompensas.

Sabiendo que el proceso que se está modelando no es un proceso determinista, de la ecuación original de Bellman se modifica los valores $V(s')$ ya que cada decisión tiene una cierta probabilidad, entonces se desea encontrar el valor esperado de tomar una cierta decision s' es decir, se quiere encontrar el valor: $\sum_{s'}^{S} P(s, a, s') * V(s')$, el cuál es la suma de los valores furutos $V(s')$ ponderados por la probabilidad de llegar al estado s' tomando la acción a, desde en estado s.

Por lo que, considerando lo antes mencionado en la ecuación inicial se

tiene:

$$V(s) = \max_{a}[R(s,a) + \gamma \sum_{s'}^{S} P(s,a,s') * V(S')]$$

Es decir, la ecuacion incial se modifica considerando la esperanza de $V(s')$, así:

$$V(s) = \max_{a}[R(s,a) + \gamma E[V(s')]]$$

2.3. Factor de penalización

Esta técnica implica la aplicación de una penalización o castigo constante a cada paso de tiempo que el agente toma mientras interactúa con su entorno.(5) La idea detrás de la penalización de vida es incentivar al agente a alcanzar su objetivo lo más rápido posible, evitando así comportamientos indeseados que prolonguen innecesariamente el proceso de aprendizaje.

Por otro lado, es importante considerar el impacto de diferentes tipos de recompensas en el aprendizaje por refuerzo. Las recompensas positivas y negativas, junto con las recompensas iguales, desempeñan roles distintos en la motivación y el comportamiento del agente. Mientras que las recompensas positivas pueden incentivar al agente a repetir ciertos comportamientos, las recompensas negativas pueden desalentar comportamientos no deseados. En algunos casos, todas las acciones pueden tener la misma recompensa, lo que puede influir en la exploración y la explotación del agente.(7)

Finalmente, es crucial considerar cómo la relación entre la recompensa total y la ganancia en la meta puede afectar el comportamiento del agente. Si la recompensa inmediata es significativamente mayor que la ganancia en la meta a largo plazo, el agente puede verse tentado a buscar gratificaciones instantáneas en lugar de perseguir objetivos a largo plazo. Esto puede conducir a comportamientos subóptimos y afectar el proceso de aprendizaje del agente en su conjunto.

2.4. Q-Learning

El Q-Learning es un algoritmo fundamental, que se utiliza para aprender una política óptima de comportamiento en un entorno desconocido y estocástico. La esencia del Q-Learning radica en la estimación de la función de valor de acción óptima, conocida como Q-valor, para cada par de estado-acción posible en el entorno. La función de valor $Q(s, a)$ representa el valor esperado acumulado que se obtiene al tomar la acción a en el estado s y luego seguir una política óptima.

Para comprender mejor el Q-Learning, es esencial entender la actualización de los valores Q utilizando la ecuación de Bellman. La ecuación de Bellman para la función de valor $Q(s, a)$ se define como:

$$Q(s, a) = R(s, a) + \gamma \cdot \max_{a'} Q(s', a')$$

La ecuación de Bellman establece que el valor Q para un par estado-acción debe ser igual a la recompensa inmediata más el valor Q esperado del mejor próximo estado y acción.(6) La actualización de los valores Q se realiza iterativamente a medida que el agente interactúa con el entorno y recibe retroalimentación en forma de recompensas.

El algoritmo Q-Learning utiliza una estrategia de exploración y explotación para aprender de manera eficiente la política óptima. Durante la fase de exploración, el agente toma acciones aleatorias para explorar el espacio de estados y acciones. Durante la fase de explotación, el agente elige las acciones que tienen el mayor valor Q estimado para el estado actual.

Es decir, se refiere a la evaluacion de la calidad de las acciones que se podría tomar estando en un estado determinado.

2.5. Diferencial Temporal

El diferencial temporal TD, es una técnica fundamental que permite actualizar las estimaciones de los valores de acciones o estados en función de la nueva información recibida a través de la interacción del agente con su entorno. Este enfoque se basa en la premisa de que la estimación actualizada de un valor debería ser una combinación ponderada del valor anterior y la nueva información obtenida a través de las recompensas y transiciones de estado.(8)

La actualización se formula mediante la ecuación de Bellman, que relaciona el valor de una acción o estado con la recompensa inmediata y el valor esperado de las futuras recompensas. En su forma más general, la actualización se expresa como:

$$Q(s,a) = Q(s,a) + \alpha \cdot (TD)$$

El diferencial temporal TD, por su parte, se calcula como la diferencia entre la recompensa obtenida después de tomar la acción y la estimación anterior del valor de esa acción, ajustada por el valor esperado de las futuras recompensas. Matemáticamente, se define como:

$$TD = R(s,a) + \gamma \cdot Q(s',a') - Q(s,a)$$

Al introducir el concepto del tiempo en esta parte del desarrollo se puede ver este concepto como una serie de tiempo de la siguiente manera:

$$TD_t = R_t(s,a) + \gamma \cdot Q_t(s',a') - Q_{t-1}(s,a)$$

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha \cdot (TD)_t$$

2.5.1. Tasa de aprendizaje α

El valor de α, también conocido como tasa de aprendizaje, es un parámetro crítico en los algoritmos de aprendizaje por refuerzo, especialmente

en el contexto del diferencial temporal TD. Su función principal es determinar la rapidez con la que se actualizan los valores de estimación de las acciones o estados (Q-valores) en función de la nueva información recibida.

El valor de α juega un papel fundamental en la convergencia y estabilidad del proceso de aprendizaje. Si α es demasiado pequeño, las actualizaciones de los Q-valores serán muy graduales, lo que puede llevar a un aprendizaje lento y a que el agente necesite más iteraciones para converger hacia una solución óptima.(3) Por otro lado, si α es demasiado grande, las actualizaciones serán muy agresivas y podrían oscilar en exceso, lo que podría dificultar la convergencia del algoritmo o incluso hacer que diverja.

En esencia, la elección adecuada de α es crucial para lograr un equilibrio entre la estabilidad y la velocidad de convergencia del algoritmo de aprendizaje.(1) Es importante ajustar cuidadosamente el valor de α durante el proceso de entrenamiento del agente, mediante experimentación y análisis empírico, para encontrar el valor óptimo que permita alcanzar una convergencia rápida.

2.6. Deep Learning

2.7. Deep Q-Learning

2.8. Redes Neuronales Convolucionales (CNN)

Capítulo 3

Metodología

3.1. Entorno de desarrollo y herramientas utilizadas

En esta sección, describiremos las tecnologías, herramientas y librerías que serán empleadas en el desarrollo del proyecto. La elección adecuada de estas tecnologías es crucial para garantizar la eficiencia y efectividad del trabajo realizado.

3.1.1. Lenguaje de programación

Se ha elegido el lenguaje de programación Python 3.7 debido a su popularidad, versatilidad y amplio soporte en el ámbito de la inteligencia artificial y el aprendizaje automático. Su flexibilidad permite integrar diferentes componentes y tecnologías, mientras que su escalabilidad y rendimiento hacen posible la implementación eficiente de modelos de RL incluso en grandes conjuntos de datos.

Por otro lado, la versión elegida se debió a la compatiblilidad con las demás librarias a instalar.

3.1.2. Librerias y Frameworks

En cuanto a las librerías y frameworks, utilizaremos TensorFlow y PyTorch para el desarrollo de modelos de aprendizaje profundo. TensorFlow es una biblioteca de aprendizaje automático de código abierto desarrollada por Google, que ofrece una amplia gama de herramientas y recursos para la construcción y entrenamiento de modelos de aprendizaje automático y redes neuronales. Por otro lado, PyTorch es una biblioteca de aprendizaje automático de código abierto respaldada por Facebook, que se destaca por su flexibilidad y facilidad de uso, especialmente en el desarrollo de modelos de aprendizaje profundo.

Además, haremos uso de la librería AIgym, que ofrece una amplia variedad de entornos de simulación y herramientas para evaluar el rendimiento de los algoritmos de aprendizaje por refuerzo en tareas de control y toma de decisiones.

3.1.3. Entorno de Desarrollo Integrado (IDE)

Como entorno de desarrollo integrado (IDE), optaremos por Visual Studio Code, el cual, proporciona un entorno de desarrollo robusto que facilita la escritura de código, la visualización de datos y la documentación del mismo. Su interfaz intuitiva y personalizable nos permitirá realizar experimentos de manera eficiente y comprender mejor el proceso de investigación. Además, su integración con herramientas de control de versiones como Git nos ayudará a mantener un seguimiento del progreso del proyecto.

3.2. Algoritmo RL básico

Para comprender el concepto fundamental del Aprendizaje por Refuerzo (RL), es crucial entender cómo un agente aprende de las acciones que toma en su entorno mientras avanza en él. Un punto de partida es examinar cómo funcionaría un agente que toma decisiones al azar y qué consecuencias acarrearía para dicho agente.

El siguiente algoritmo utiliza el entorno de apendizaje llamado "$MountainCar-v0$" de la libreria GYM (Anexo A.1):

1. **Iniciar:**
   - Importar la herramienta de simulación Gym.
   - Crear un entorno de simulación llamado "MountainCar-v0".
   - Definir el número máximo de intentos de aprendizaje.
2. **Para cada intento de aprendizaje:**
   - Reiniciar el entorno para comenzar desde el principio.
   - Establecer la recompensa total y el número de pasos en cero.
3. **Mientras el intento de aprendizaje no esté completo:**
   - Mostrar el entorno de simulación.
   - Tomar una decisión aleatoria de movimiento para el coche.
   - Ejecutar la decisión tomada y observar el resultado.
   - Actualizar la recompensa total y el número de pasos.
4. **Imprimir el resultado del intento de aprendizaje:**
   - Mostrar el número de pasos realizados.
   - Mostrar la recompensa total obtenida.
5. **Repetir el proceso hasta que se completen todos los intentos de aprendizaje.**
6. **Cerrar el entorno de simulación al finalizar.**

El análisis del algoritmo revela la ausencia de cualquier forma de retroalimentación del agente. En otras palabras, el agente no incorpora información de las acciones tomadas en cada estado mientras avanza.

3.3. Classe QLearner

Como se ha mencionado previamente, al agente le resulta crucial considerar las probabilidades asociadas a cada acción antes de decidir cuál sería la mejor elección. Por esta razón, se hace necesario implementar la clase QLearner. Esta clase le proporcionará al agente las herramientas necesarias para aprender a evaluar y seleccionar entre las diferentes acciones disponibles. Para alcanzar este objetivo, se deben incorporar las siguientes funciones dentro de la clase QLearner. (Anexo A.1)

3.3.1. Función $init$:

En esta función se inicializará el objeto $self$, el cual, a través de diferentes métodos, almacenará características esenciales del entorno. Entre ellas se incluyen: el tamaño del espacio de estados, el valor máximo dentro del espacio de estados, el tamaño del espacio de acciones, el factor de descuento (gamma) y la tasa de aprendizaje (alpha). Además, se llevará a cabo la discretización del espacio de acciones, una operación que puede variar dependiendo de las especificidades de cada entorno.

3.3.2. Función $discretize$:

En esta función se implementa de tal forma que se pueda determinar en qué discretización se encuentra el estado actual del agente. Esto es crucial para que el agente pueda interpretar y tomar decisiones basadas en su entorno.

3.3.3. Función $get\ action$:

Aquí es donde el agente tomará las decisiones basadas en las probabilidades de éxito (ϵ) o fracaso $(1-\epsilon)$. Así, como política de fuerza bruta se elige un epsilon mínimo, el cual será el valor que va aprendiendo mientras el incremento del aprendizaje $self.epsilon$ sea superior a ese valor, por lo que se elige la acción en base a este criterio. Por el contrario, si este criterio no se cumple, el agente tomará acciones aleatorias hasta poder

cumplir con el criterio anterior. Esto es posible realizando un decremento en $self.epsilon$ de acuerdo con el epsilon mínimo elegido y el número máximo de pasos a realizarse en todos los episodios.

3.3.4. Función $learn$:

En esta etapa de la implementación, el agente tiene la capacidad de aprender a partir de las acciones tomadas en el entorno en estados específicos. Aquí es donde se codifica la ecuación del diferencial temporal. Esta ecuación permite construir una matriz de calidad, que servirá como guía para que el agente pueda tomar decisiones más informadas y mejorar su desempeño a lo largo del tiempo, esta matriz refleja la estimación de la recompensa esperada para cada acción en cada estado, lo que ayuda al agente a elegir las acciones que maximizan su recompensa a largo plazo. Además, durante este proceso de aprendizaje, el agente ajusta gradualmente sus estimaciones de calidad a medida que explora y experimenta el entorno.

3.4. Entrenamiento QLearner

Con la clase Qlearner ya implementada, se procede a construir la función que utilizará sus métodos para actualizar la matriz Q. Esta actualización permite al agente tomar mejores decisiones a lo largo del tiempo, así como desarrollar una política eficiente para enfrentarse al entorno después de varios episodios de aprendizaje. La matriz Q es esencialmente una tabla que asigna un valor a cada par de estado-acción, representando la calidad estimada de tomar una acción en un estado específico. Al actualizar esta matriz utilizando la ecuación de actualización Q-learning, el agente puede aprender de su experiencia y mejorar su rendimiento en el entorno. Este proceso de aprendizaje iterativo permite al agente adaptarse y tomar decisiones más informadas a medida que interactúa con el entorno. (Anexo A.1)

Algoritmo de Entrenamiento del Agente

1. Se define una función llamada "train". Esta función recibe como argumentos el agente a entrenar y el entorno en el que va a aprender.

2. El bucle principal del entrenamiento se realiza a lo largo de un número predeterminado de episodios. En cada episodio:

   a) Se restablece el entorno a su estado inicial y se comienza a interactuar con él.

   b) El agente selecciona acciones en base a su estrategia de aprendizaje, utilizando la ecuación Q-learning para tomar decisiones.

   c) Después de ejecutar una acción:

      - Se observa el siguiente estado.
      - Se recibe una recompensa.
      - Se actualiza el conocimiento del agente sobre el entorno a través de un proceso de aprendizaje.

   d) El bucle continúa hasta que se alcanza un estado final en el episodio.

3. Al final de cada episodio:

   - Se acumula la recompensa total obtenida.
   - Se registra la mejor recompensa obtenida hasta el momento.
   - Se imprime información relevante, como el número de episodio, la recompensa total y la mejor recompensa alcanzada.

4. Al finalizar todos los episodios de entrenamiento:

   - Se devuelve la mejor política de acción aprendida por el agente, que es aquella que maximiza la estimación de calidad de acciones para cada estado.

5. Este proceso de entrenamiento permite al agente mejorar gradualmente su desempeño en el entorno, aprendiendo a tomar decisiones más efectivas y maximizando su recompensa total a lo largo del tiempo.

Capítulo 4

Resultados

Capítulo 5

Conclusiones y recomendaciones

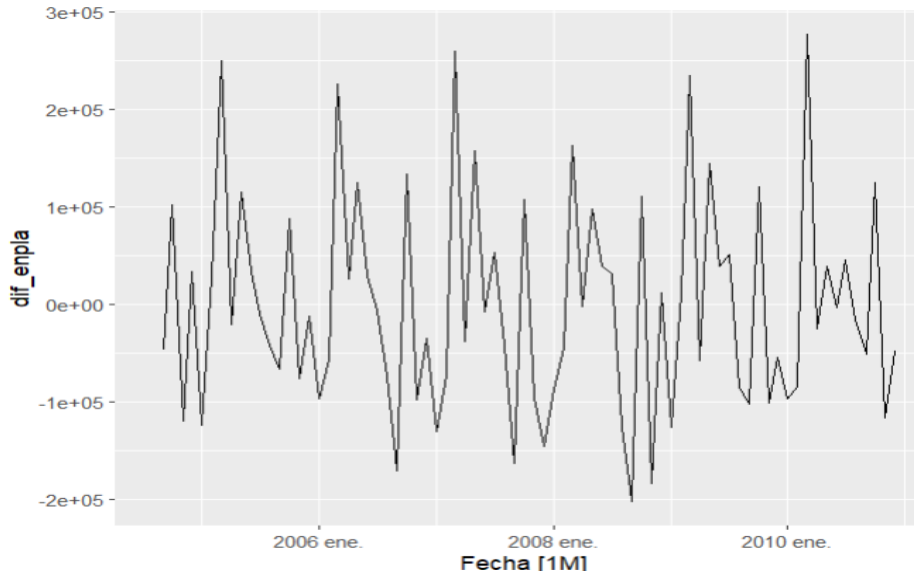
5.1. Conclusiones

5.2. Recomendacione

Capítulo A

Anexos

A.1. Código del algoritmo RL básico



Referencias bibliográficas

[1] Wolfram Barfuss, Jonathan F. Donges, and Jürgen Kurths. Deterministic limit of temporal difference reinforcement learning for stochastic games. 2019. URL: https://www.nature.com/articles/s41598-019-56406-5.

[2] Richard Ernest Bellman. The theory of dynamic programming, Oct 2008. URL: https://www.rand.org/content/dam/rand/pubs/papers/2008/P550.pdf.

[3] Kristopher De Asis, Alan Chan, Silviu Pitis, Richard S. Sutton, and Daniel Graves. Fixed-horizon temporal difference methods for stable reinforcement learning. 2019. URL: https://arxiv.org/abs/1907.04402.

[4] Arthur Juliani. Simple reinforcement learning with tensorflow part 0: Q-learning with tables and neural networks, Aug 2016. URL: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-.

[5] Volodymyr Mnih et al. Asynchronous methods for deep reinforcement learning. 2016. URL: https://spinningup.openai.com/en/latest/spinningup/keypapers.html.

[6] John Schulman et al. High-dimensional continuous control using generalized advantage estimation. 2015. URL: https://spinningup.openai.com/en/latest/spinningup/keypapers.html.

[7] John Schulman et al. Trust region policy optimization. 2015.

URL: https://spinningup.openai.com/en/latest/spinningup/keypapers.html.

[8] William Uther. Temporal difference learning. 1988. URL: https://www.ijcai.org/Proceedings/88-1/Papers/122.pdf.