



ugr

Universidad
de Granada

TRABAJO FIN DE MÁSTER

MÁSTER DE CIENCIA DE DATOS E INGENIERÍA DE COMPUTADORES

Uso de Modelos Generativos en Aprendizaje por Refuerzo

Autora

Silvia Barroso Moreno

Directores

Juan Gómez Romero

Miguel Molina Solana



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, 6 de septiembre de 2023



Uso de modelos generativos en aprendizaje por refuerzo

Autora

Silvia Barroso Moreno

Directores

Juan Gómez Romero

Miguel Molina Solana

Yo, **Silvia Barroso Moreno**, alumna de la titulación Máster Universitario Oficial en Ciencia de Datos e Ingeniería de Computadores de la Universidad de Granada, con DNI 49616461X, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca de la universidad para que pueda ser consultada por las personas que lo deseen.

Fdo: Silvia Barroso Moreno

Granada, a 6 de Septiembre de 2023

D. **Juan Gómez Romero**, Catedrático del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada

D. **Miguel Molina Solana**, Profesor Titular del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada

INFORMAMOS:

Que el presente trabajo, titulado ***Uso de modelos generativos en aprendizaje por refuerzo***, ha sido realizado bajo nuestra supervisión por **Silvia Barroso Moreno**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expedimos y firmamos el presente informe en Granada a 6 de Septiembre de 2023.

Los directores:

Juan Gómez Romero

Miguel Molina Solana

Índice general

Agradecimientos	I
Resumen	III
Summary	V
Introducción	VII
1. Motivación	VII
2. Aproximación	VIII
3. Objetivos	X
4. Estructura	XI
I. Fundamentos teóricos	1
1. Aprendizaje por Refuerzo	3
1.1. Fundamentos	3
1.2. Métodos para soluciones tabulares	5
1.2.1. Métodos acción-valor	5
Función acción-valor	6
Implementación incremental de la función acción-valor	7
Selección de acciones con un límite de confianza superior	8
1.2.2. Procesos de decisión finitos de Markov	8
Formalización Matemática MDPs	8
Políticas y funciones de valor	9
Ecuaciones de Bellman	10
Programación dinámica	11
1.2.3. Métodos de Monte Carlo	11
Predicción con Monte Carlo	12
Control con Monte Carlo	12
1.2.4. Aprendizaje de diferencia temporal	12
Predicción TD	13
Control On-Policy TD: SARSA	13
Control Off-Policy TD: Q-Learning	13
Expected SARSA	14
1.2.5. n-steps Bootstrapping	14
Predicción: n-step TD	14
Control: n-step SARSA	14

1.2.6.	Resumen	15
1.3.	Métodos para soluciones aproximadas	15
1.3.1.	Parametrización de la función valor	16
	Métodos del gradiente estocástico	16
	Métodos de semi-gradientes	17
	El gradiente de Monte Carlo con agregación de estados	17
	Métodos lineales	17
1.3.2.	Parametrización de la función acción-valor	18
	Episodic SARSA con aproximación de funciones	18
	Expected SARSA con aproximación de funciones	19
	La recompensa promedio	19
1.3.3.	Métodos de gradientes de políticas	19
	Aprendiendo políticas parametrizadas	20
	Gradiente de políticas	20
	Actor-Crítico para tareas continuas	21
1.4.	Algoritmos de Aprendizaje Profundo por Refuerzo	22
1.4.1.	DQN	22
1.4.2.	A2C	24
1.4.3.	DDPG	25
1.4.4.	TRPO	27
1.4.5.	PPO	29
1.4.6.	SAC	31
2.	Redes Generativas Adversarias (GANs)	33
2.1.	Fundamentos	33
2.1.1.	Concepto	33
2.1.2.	Funcionamiento	34
2.2.	Entrenamiento de GANs	35
2.2.1.	Funciones de coste	38
2.2.2.	Dificultades del aprendizaje	40
2.3.	Fundamentos matemáticos de las GANs	40
2.3.1.	Enfoque básico de las Redes Generativas Adversarias (GANs) . . .	41
2.3.2.	Problema min-max	42
2.3.3.	Ejemplo de problema min-max	43
2.3.4.	Formulación min-max del enfoque básico de las GANs	44
2.3.5.	Medidas de similitud entre dos distribuciones de probabilidad . . .	45
	f-divergencia	46
2.3.6.	Solución al problema minmax	48
2.3.7.	Algoritmo de entrenamiento de las Redes Generativas Adversarias (GANs)	48
3.	Aprendizaje por Imitación	51
3.1.	Fundamentos	51

3.2. Aprendizaje por Imitación Generativo Adversario (GAIL)	52
3.2.1. Concepto	52
3.2.2. Aprendizaje por Refuerzo Inverso (IRL)	54
3.2.3. Medidas de ocupación	55
3.2.4. Conexión Aprendizaje por Imitación y GANs	56
RL seguido de IRL	57
Aprendizaje por Imitación Generativo Adversario	57
¿Dónde está la conexión con las GANs?	58
3.2.5. Algoritmo GAIL	59
4. Hibridación Q-Learning (HQL)	61
4.1. Fundamentos	61
4.2. Una perspectiva distribucional del aprendizaje por refuerzo	62
4.3. Formulación	63
4.4. Algoritmo de entrenamiento HQL	64
II. Experimentación y resultados	67
5. Métodos y herramientas	69
5.1. Metodología	69
5.2. Implementación	71
5.2.1. Implementación GAIL	71
Algoritmo de Aprendizaje por Refuerzo: PPO	74
Modelo generativo y de aprendizaje por imitación: GAIL	75
Particularidades de la configuración GAIL en Sinergym	78
5.2.2. Implementación HQL	79
Modelo generativo: GANs	82
5.3. Sinergym	83
6. Experimentación con entornos de OpenAI GYM	89
6.1. Entorno de ejecución	89
6.2. Descripción	90
6.3. CartPole	90
6.3.1. Configuración	91
6.3.2. Ejecución	91
6.3.3. Resultados	92
6.4. Taxi v3	96
6.4.1. Configuración	97
6.4.2. Ejecución	97
6.4.3. Resultados	97
Experimento 1: HQL	97
Experimento 2: GAIL	101

7. Experimentación con entornos de Sinergym	105
7.1. Entorno de ejecución	105
7.2. Descripción	106
7.3. 5Zone	107
7.3.1. Configuración	107
7.3.2. Ejecución	107
7.3.3. Resultados	107
7.4. Datacenter	111
7.4.1. Configuración	111
7.4.2. Ejecución	111
7.4.3. Resultados	111
7.5. Warehouse	114
7.5.1. Configuración	114
7.5.2. Ejecución	114
7.5.3. Resultados	115
8. Discusión	119
8.0.1. Resumen de resultados	119
8.1. Comparativa HQL vs GAIL en Taxi	120
8.2. Comparativa GAIL vs PPO	120
III. Conclusiones y trabajo futuro	123
9. Conclusiones y Trabajo Futuro	125
A. Uso de Decision Diffuser en Aprendizaje por refuerzo	127
A.1. Introducción	127
A.1.1. Aprendizaje por refuerzo offline	128
A.1.2. Modelos de difusión probabilísticos	128
Difusión guiada o libre	129
A.2. Modelado generativo con Decision Diffuser	129
A.2.1. Estados y acciones	130
Dinámica inversa	130
A.2.2. Planificación con orientación un clasificador libre	130
A.2.3. Condicionamiento mas allá del retorno	131
A.2.4. Entrenamiento de Decision Diffuser	131
Función de pérdida	132
Arquitectura	132

Índice de figuras

1.1. Funcionamiento de los algoritmos RL	3
1.2. Agente y entorno [ZB20]	4
1.3. Ejemplo de función acción-valor	6
1.4. Esquema del aprendizaje por refuerzo	9
2.1. Funcionamiento de una GANs [LB19]	34
2.2. Entrenamiento de una GANs [LB19]	36
2.3. Ejemplo MinMax f	43
3.1. Algoritmos de Aprendizaje por Imitación[Sew19]	53
5.1. Esquema GAIL	71
5.2. Esquema HQL: aplicamos una GAN a una base de datos experta de tablas Q-Learning	79
5.3. HQL: Red neuronal convolucional del discriminador	80
5.4. Edificio 5Zone [Sina]	84
5.5. Edificio Datacenter [Sina]	85
5.6. Edificio Warehouse [Sina]	87
6.1. Entorno clásico de control <i>CartPole</i> [TTK ⁺ 23]	91
6.2. Cartpole: Evaluación del Generador GAIL en las épocas 10, 20, 30, 40 (Eje x-Nº episodio, Eje y-Recompensa)	93
6.3. Cartpole: Evaluación del Generador GAIL en las épocas 50, 60, 70, 80, 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)	94
6.4. Evaluación del Generador GAIL Cartpole: épocas-Recompensa media	95
6.5. Entorno de Toy Text: Taxi [TTK ⁺ 23]	96
6.6. Taxi: Evaluación del Generador HQL en las épocas 10, 20, 30, 40, 50, 60, 70, 80 (Eje x-Nº episodio, Eje y-Recompensa)	99
6.7. Taxi: Evaluación del Generador HQL en las épocas 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)	100
6.8. Evaluación del Generador HQL Taxi: épocas-Recompensa Media	100
6.9. Taxi: Evaluación del Generador GAIL en las épocas 10, 20, 30, 40, 50, 60 (Eje x-Nº episodio, Eje y-Recompensa)	102
6.10. Taxi: Evaluación del Generador GAIL en las épocas 70, 80, 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)	103
6.11. Evaluación del Generador GAIL Taxi: épocas-Recompensa Media	104

7.1. 5Zone: Evaluación del Generador en las épocas 10, 20, 30, 40, 50, 60 (Eje x-Nº episodio, Eje y-Recompensa)	109
7.2. 5Zone: Evaluación del Generador en las épocas 70, 80, 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)	110
7.3. Evaluación del Generador GAIL 5Zone: épocas-Recompensa Media	110
7.4. Datacenter: Evaluación del Generador en las épocas: 10, 20 (Eje x-Nº episodio, Eje y-Recompensa)	112
7.5. Datacenter: Evaluación del Generador en las épocas: 30, 40, 50, 60, 70, 80, 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)	113
7.6. Comparativa Datacenter	114
7.7. Warehouse: Evaluación del Generador en las épocas: 10, 20, 30, 40, 50, 60, 70, 80 (Eje x-Nº episodio, Eje y-Recompensa)	116
7.8. Warehouse: Evaluación del Generador en las épocas: 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)	117
7.9. Comparativa Warehouse	117
8.1. Comparativa Taxi: GAIL vs HQL	120
8.2. Comparativa Taxi: GAIL vs HQL vs PPO	121
A.1. Esquema Decision Diffuser	127
A.2. Planificación en Decision Diffuser	129

Índice de tablas

5.1. Experimentación	70
5.2. Acciones 5Zone	85
5.3. Observaciones 5Zone	85
5.4. Observaciones Datacenter	86
5.5. Acciones Warehouse	87
5.6. Observaciones Warehouse	87
6.1. Cartpole: Entrenamiento GAIL	92
6.2. Cartpole: Evaluación Discriminador GAIL	93
6.3. Taxi: Entrenamiento HQL	98
6.4. Taxi: Evaluación Discriminador HQL	98
6.5. Taxi: Entrenamiento GAIL	101
6.6. Taxi: Evaluación Discriminador GAIL	101
7.1. 5Zone: Entrenamiento GAIL	108
7.2. 5Zone: Evaluación Discriminador	108
7.3. Datacenter: Entrenamiento GAIL	112
7.4. Datacenter: Evaluación Discriminador	112
7.5. Warehouse: Entrenamiento GAIL	115
7.6. Warehouse: Evaluación Discriminador	115
8.1. Comparativa global: Recompensas medias / Entornos	119

Agradecimientos

A mi familia, quienes me han brindado su apoyo incondicional día tras día. Gracias por proporcionarme los mejores consejos y por ser mi fuente de ánimo en cada adversidad. Sin ellos, nada de esto sería posible. Son el pilar fundamental que me impulsa a perseguir todos mis sueños. Ellos han estado respaldándome para que nunca desista y para que busque reconocer las señales que me guíen hacia este camino que se ha convertido en mi norte y guía.

A mis estimados tutores, Juan y Miguel, a los que les agradezco profundamente su constante atención durante todo este trabajo. Gracias, por formarme en un ámbito que me apasiona y por introducirme en un mundo desconocido, pero a la vez fascinante. Agradecer a Alejandro Campoy todo su apoyo en este trabajo, con sus explicaciones, correcciones y su gran aporte de la herramienta Sinergym. Gracias a Juan, por confiar de nuevo en mí. Solo tengo palabras de gratitud hacia ellos.

A mis amigos y compañeros, quienes me han irradiado felicidad y tranquilidad en todo este proceso. Gracias por creer en mí y por sacar lo mejor de mí en los momentos difíciles.

Gracias a mi amiga Pilar, siempre me transmitió su perseverancia y constancia ante cualquier objetivo,

Gracias a mis amigas Marimar y Juana Mari junto con mi hermana Noelia, mi gran apoyo desde Medina Sidonia (Cádiz) quienes se han interesado constantemente por mi trabajo y me han aportado todos sus ánimos.

Gracias a mis compañeros, Eusebio, Chus y Álvaro, y todos los demás, quienes diariamente han alegrado mis días de trabajo con desayunos en la cafetería y almuerzos acompañados de altas dosis de amistad y compañerismo.

Gracias a Ana y Celia, son las mejores compañeras de piso y me infundieron toda su determinación ante sus sueños, me transmitieron que toda meta era posible.

A cada una de las personas del grupo de SAIL y del CITIC, quienes me han acogido con los brazos abiertos y han aportado su granito de arena. Gracias por cada sonrisa que me han transmitido, por infundirme la determinación para seguir adelante.

A Granada porque sigue conquistando a esta gaditana quien comparte su corazón entre la bahía luminosa de su Cádiz y el atardecer mágico de Granada, entre su familia gaditana y asidonense y sus años de formación en Granada.

Resumen

Uso de modelos generativos en aprendizaje por refuerzo

Silvia Barroso Moreno

Palabras clave: Redes Generativas Adversarias, GANs, Aprendizaje por Profundo por Refuerzo, DRL, problemas de control, experto, GAIL, Generative Adversarial Imitation Learning, HQL, Hibridación Q-Learning, OpenAI Gym, Sinergym, Decision Diffuser

Resumen

Este Trabajo de Fin de Máster (TFM) investiga la confluencia de las Redes Generativas Adversarias (GANs) con el Aprendizaje por Profundo por Refuerzo (DRL) para solucionar problemas de control en sistemas autónomos. El trabajo aborda dos líneas de investigación fundamentales. La primera explora el algoritmo GAIL (*Generative Adversarial Imitation Learning*), que utiliza GANs para sintetizar políticas de decisión que imitan a expertos, basándose en históricos de secuencias estado-acción. La segunda introduce un nuevo algoritmo denominado HQL (*Hibridación Q-Learning*), que es una extensión del algoritmo Q-Learning para incluir datos históricos en el entrenamiento con el entorno. Este nuevo enfoque se centra en generar tablas sintéticas $Q(s, a)$ (valor aproximado de la recompensa para un estado y acción) para entrenar agentes de DRL, imitando la estimación de la recompensa futura en lugar de la propia política. Antes de abordar estos algoritmos, se lleva a cabo un análisis profundo de los fundamentos teóricos y matemáticos detrás de DRL y GANs, así como una revisión detallada de literatura existente.

Para validar los algoritmos investigados, se ha llevado a cabo una experimentación en entornos simplificados y en aplicaciones del mundo real. Así, se utilizan entornos estándar de OpenAI Gym para evaluar la eficacia de los algoritmos en tareas de control bien definidas como *CartPole* y *Taxi*. Además, se aplica el algoritmo GAIL en un contexto más práctico, resolviendo problemas de control de energía en edificios a través del software Sinergym. Las pruebas empíricas en estos entornos proporcionan una primera evaluación de la aplicabilidad de GAIL e HQL en escenarios variados, destacando la potencial utilidad de HQL. Para finalizar, el trabajo identifica líneas de investigación futuras, como es la incorporación de otros modelos generativos como *Decision Diffuser*.

Summary

Use of generative models in reinforcement learning

Silvia Barroso Moreno

Keywords: Generative Adversarial Networks, GANs, Deep Reinforcement Learning, DRL, control problems, expert, GAIL, Generative Adversarial Imitation Learning, HQL, Hybridization Q-Learning, OpenAI Gym, Sinergym, Decision Diffuser

Abstract

This Master's Thesis investigates the convergence of Generative Adversarial Networks (GANs) with Deep Reinforcement Learning (DRL) to solve control problems in autonomous systems. The work addresses two fundamental lines of research. The first explores the GAIL algorithm (*Generative Adversarial Imitation Learning*), which uses GANs to synthesize decision-making policies that imitate experts, based on historical state-action sequences. The second introduces a new algorithm called HQL (*Hybridization Q-Learning*), which is an extension of the Q-Learning algorithm that uses historical data to enrich the training process with the environment. This new approach focuses on generating synthetic $Q(s, a)$ tables (approximate value of the reward for a state and action) to train DRL agents, imitating the estimation of future rewards rather than the policy itself. Before tackling these algorithms, a thorough analysis of the theoretical and mathematical foundations behind DRL and GANs is carried out, as well as a detailed review of existing literature.

To validate the investigated algorithms, experimentation has been conducted in simplified environments and real-world applications. Standard OpenAI Gym environments are used to evaluate the efficacy of the algorithms in well-defined control tasks such as *CartPole* and *Taxi*. Additionally, the GAIL algorithm is applied in a more practical context, solving energy control problems in buildings through the Sinergym software. Empirical tests in these environments provide an initial assessment of the applicability of GAIL and HQL in varied scenarios, highlighting the potential utility of HQL. To conclude, the work identifies future lines of research, including the incorporation of other generative models such as *Decision Diffuser*.

Introducción

Este capítulo describe la motivación y los objetivos del trabajo fin de máster, así como la estructura de la memoria. Pretende asimismo contextualizar el aprendizaje por refuerzo y los modelos generativos y destacar su importancia dentro del aprendizaje automático y la inteligencia artificial.

1. Motivación

El término de *refuerzo* fue introducido por Pavlov a principios del siglo XX mientras investigaba la psicología y la psicopatología de los animales mediante estímulos y respuestas condicionadas. Por ejemplo, uno de los experimentos que realizó consistía en hacer sonar una campana justo antes de darle la comida a un perro con el objetivo de observar su comportamiento. Pavlov descubrió que, tras ejecutar la acción de sonar la campana, siempre ocurría que el perro salivaba si a continuación le daba la recompensa de la comida. Realizó mas experimentos de condicionamiento con recompensas y descubrió que la fuerza de un comportamiento del animal se modifica con la recompensa o el castigo y que el comportamiento depende de los estímulos que desencadenan la respuesta.

Las ideas de Pavlov influyeron en numerosos ámbitos científicos, incluyendo la Computación. Así, la teoría de control óptimo aparece en 1954 cuando Bellman introdujo el concepto de función de valor [SB18], que representa la estimación de la recompensa futura de un agente inteligente. En este marco, la resolución de problemas de optimización se formaliza como un proceso de decisión de Markov (*Markov Decision Process*, *MDP*) y se resuelve aplicando programación dinámica. Basados en estos hallazgos, en 1968 aparecen los primeros algoritmos de aprendizaje por refuerzo (*reinforcement learning*, *RL*), que realizan experimentos de prueba y error sobre un entorno para descubrir políticas de actuación óptimas. En la década de los 70, el RL comienza a aplicarse más intensamente a problemas de juegos y optimización, como el problema de la máquina tragaperras de n brazos (*N-armed bandit problem*).

El aprendizaje por refuerzo profundo (*deep reinforcement learning*, *DRL*) es una extensión del RL que utiliza redes neuronales (*deep learning*, *DL*) como aproximador de alguna de las funciones del problema de RL, como la función de valor o la política actuación. El DRL se dio a conocer a principios de la década pasada [MKS⁺15] cuando le empresa DeepMind desarrolló un algoritmo capaz de jugar a una variedad de juegos de Atari (*deep Q-network*, *DQN*) solamente a partir de imágenes. Los intentos anteriores de crear agentes para resolver estos juegos implicaban procesar las observaciones para extraer información útil en la toma de decisiones (ingeniería de características) e incluir

conocimiento experto sobre las reglas específicas del juego. Estos enfoques pueden funcionar bien para un juego en particular, pero son difíciles de extender a distintos juegos y, sobre todo, a diferentes dominios. Así, en los últimos años el DRL ha demostrado su utilidad en numerosas aplicaciones, como el control de robots, la planificación de rutas, la administración de recursos o el control industrial, entre otros.

Por otro lado, recientemente se ha producido un enorme auge de los modelos generativos de aprendizaje automático [Fos19], que persiguen crear nuevos contenidos (textos, imágenes, vídeo, etc.) coherentes con los ejemplos proporcionados para su entrenamiento. Avances recientes como Midjourney o Dall-E para imágenes o ChatGPT para texto ilustran la capacidad de los algoritmos de aprendizaje generativo para producir datos realistas en diferentes formatos. Uno de los primeros modelos generativos en DL fue propuesto en 2014 por I. Goodfellow: las llamadas redes generativas adversarias (*generative adversarial networks*, *GANs*). Las GANs utilizan dos redes complementarias; el generador, que crea los contenidos nuevos, y el discriminador, que distingue los ejemplos proporcionados de los sintetizados por el generador. Al entrenarse ambas conjuntamente, el discriminador aprende a detectar datos artificiales, mientras que el generador mejora progresivamente su capacidad para engañar al discriminador. Es interesante destacar que las GANs no necesitan datos etiquetados, solo ejemplos de los contenidos que se quieren imitar.

En este trabajo fin de máster se estudia cómo utilizar GANs para generar contenido sintético que pueda ser usado por agentes en un contexto de aprendizaje por refuerzo. La hipótesis de partida es que sería posible utilizar datos sintéticos generados por GANs u otros modelos para recabar experiencias de entrenamiento minimizando o incluso eliminando la necesidad de interactuar con el entorno. Como conjunto de entrenamiento de estos modelos generativos se usaría una base de datos de actuaciones expertas previas, que pueden provenir o no de otro agente. En consecuencia, el trabajo se sitúa dentro del marco del aprendizaje por refuerzo inverso (*inverse reinforcement learning*, *IRL*), del que se analizarán diferentes contribuciones y realizarán nuevas propuestas.

2. Aproximación

En este TFM investigamos las posibilidades de las GANs en el contexto del DRL para resolver problemas de control. Más concretamente, nos centraremos en el aprendizaje por imitación, en el que se busca aprender políticas de control mediante la imitación de las decisiones registradas en un histórico de datos, que se suponen expertas. Para ello, abordaremos dos líneas de trabajo:

1. **Estimación directa de la función que determina la política de decisión:** hemos estudiado el algoritmo GAIL (*Generative Adversarial Imitation Learning*), propuesto en [HE16]. Este algoritmo utiliza el modelo GANs para la generación sintética de una política que imita a la política experta que subyace a las secuencias sobre estados y acciones que se proporcionan como datos de entrenamiento. Esto es, tenemos una base de datos de secuencias estado-acción reales y el modelo GAN

de GAIL inventará secuencias nuevas, generadas automáticamente pero con una distribución similar a los ejemplos reales.

2. **Generación automática de datos intermedios para entrenar los agentes:** hemos propuesto un nuevo algoritmo denominado HQL (*Hibridación Q-Learning*), que extiende Q-Learning con GANs. En este caso, la GAN no genera secuencias de estados y acciones, sino las tablas $Q(s, a)$ que se calculan en Q-Learning para estimar la recompensa esperada en un estado s aplicando una acción a . A partir de las tablas Q obtenidas, reales y sintéticas, se puede extraer una política imponiendo que las recompensas medias obtenidas para cada estado s y cada acción a almacenadas en $Q(s, a)$ sean máximas.

Para validar los desarrollos realizados, realizaremos diversos experimentos utilizando los entornos estándar de OpenAI Gym [TTK⁺23] y un problema real de control de energía en edificios basado en el software Sinergym, un software creado por el grupo de investigación de los tutores [Sinb, Sina].

Para ello, en la primera etapa del trabajo estudiamos los fundamentos del DRL y de las GANs. A continuación, investigamos el paradigma del aprendizaje por imitación y, dentro de este, el algoritmo GAIL como exponente más significativo de la sinergia entre aprendizaje por refuerzo y modelos generativos. Analizamos tanto los fundamentos matemáticos en los que se basa como su desarrollo e implementación prácticas. Seguidamente, describimos el algoritmo HQL. Este algoritmo parte de un trabajo de grado previo¹, en el que utilizamos GANs para generar caras falsas (*deepfakes*) a partir de un banco de imágenes. Simplificando, en el TFG utilizamos GANs para crear matrices (imágenes sintéticas) a partir de otras matrices (imágenes reales), por lo que HQL plantea incorporar un mecanismo similar dentro del algoritmo Q-Learning 1.2.4 para generar y utilizar en el entrenamiento tablas Q sintéticas obtenidas a partir de un conjunto reducido de tablas Q reales.

En la segunda etapa del trabajo nos hemos centrado en la prueba y validación de los dos algoritmos investigados, GAIL e HQL, en problemas típicos de DRL (los entornos de OpenAI Gym) y en problemas reales de control de energía en edificios (con los edificios de Sinergym). En el primer conjunto de experimentos, GAIL se ha evaluado en los entornos *CartPole* y *Taxi*, mientras que HQL se ha evaluado solamente en *Taxi*, por tratarse de un entorno discreto donde la función Q se puede representar tabularmente. En el segundo conjunto de experimentos, utilizamos GAIL con los edificios *5Zone*, *DataCenter* y *Warehouse*.

Finalmente, como acercamiento al trabajo futuro, utilizamos otro modelo generativo moderno, *Decision Diffuser*, que ofrece algunas características que lo hacen susceptible de ser aprovechado por algoritmos DRL de igual forma que las GANs en este trabajo.

¹Silvia Barroso Moreno (2022). “Redes Generativas Adversarias para la creación de deepfakes”, <https://github.com/silviabm98/TFG>

3. Objetivos

Nuestro propósito de este trabajo es alcanzar los siguientes objetivos a través de las tareas asociadas:

1. Conocer el estado del arte del problema: Estudio, aprendizaje propio y redacción de conceptos teóricos necesarios para abordar este trabajo.
 - Identificar los fundamentos del aprendizaje profundo, del aprendizaje por refuerzo y de los modelos generativos.
 - Definir formalmente el aprendizaje por refuerzo (RL).
 - Explicar el funcionamiento e introducir los conceptos matemáticos que hay detrás de las redes generativas adversarias (GANs).
2. Conocer las técnicas de aprendizaje por imitación que utilizan las redes generativas adversarias para aprender políticas del aprendizaje por refuerzo.
 - Presentar el aprendizaje por imitación generativo adversario (GAIL), una técnica de aprendizaje por imitación donde se enlazan los conceptos previamente estudiados: RL y GANs.
 - Exponer una nueva propuesta donde se realiza una hibridación entre el aprendizaje por imitación y el aprendizaje por refuerzo, denominada Hibridación Q-Learning (HQL). Aquí se investigará la generación sintética de tablas Q a través de redes generativas adversarias.
 - Integrar modelos generativos en el proceso de entrenamiento de agentes de control.
 - Modelar el problema de control de climatización (*heating, ventilation and air-conditioning, HVAC*) en edificios del entorno Sinergym como un problema de aprendizaje profundo por refuerzo.
3. Validar los algoritmos estudiados en diversos ámbitos de aplicación de creciente complejidad: Experimentación con distintos entornos de OpenAI Gym y Sinergym.
 - Realizar experimentos con GAIL para valorar la viabilidad y conveniencia de esta aproximación anterior en el control de HVAC de la simulaciones de edificios que nos presenta el entorno Sinergym.
 - Realizar experimentos de GAIL y de HQL sobre entornos de Gym OpenAI.

4. Estructura

Nuestra presente memoria se organiza de la siguiente manera:

- En el primer capítulo, estudiaremos en profundidad el aprendizaje por refuerzo.
- En el segundo capítulo, explicaremos el modelo generativo seleccionado para desarrollar nuestra experimentación: las Redes Generativas Adversarias (GANs). Detallaremos los fundamentos teóricos y el entrenamiento de una GANs unido a la formulación matemática que presentan las GANs. Esta formulación demuestra la existencia de solución para el problema minmax y la convergencia del algoritmo de entrenamiento de una GANs. Esto es necesario, ya que veremos en el siguiente capítulo como las técnicas de aprendizaje por imitación descritas tendrán el mismo objetivo que una GAN.
- En el tercer capítulo, presentamos los fundamentos del aprendizaje por imitación y demostramos como GAIL conecta las GANs y el aprendizaje por imitación, aprendiendo a generar acciones a partir de acciones expertas, esto es, acciones consideradas buenas para un estado específico.
- En el cuarto capítulo, analizaremos como HQL permite explorar la generación sintética de la función Q-Learning.
- A partir del quinto capítulo, enfocamos todo el marco teórico estudiado en casos de estudio prácticos, realizando la experimentación de este trabajo sobre los entornos de OpenAI Gym y Sinergym.
- Finalmente, incluiremos un apéndice donde analizamos brevemente un nuevo modelo generativo, diferente a las redes generativas adversarias, conocido como Decision Diffuser.

Parte I.

Fundamentos teóricos

1. Aprendizaje por Refuerzo

En este capítulo se expondrá el estudio la revisión sobre aprendizaje por refuerzo realizado a partir de [ZB20], [SB18], [Mor20], [Sew19], [Lap18], [SLA⁺15], [Cou].

1.1. Fundamentos

El aprendizaje por refuerzo (*reinforcement learning*, *RL*) es un marco genérico para representar y resolver tareas de control. Es un tipo de aprendizaje automático en el que se mapean situaciones a partir de las acciones con el objetivo de maximizar una señal de recompensa.

La Figura 1.1 muestra cómo los algoritmos de aprendizaje por refuerzo toman acciones de tareas de control para producir recompensas, esto es, señales de refuerzo positivo o negativo, de las que se retroalimenta dicho algoritmo para realizar la siguiente acción. Las tareas de control pueden ser un juego, conducir un robot aspiradora, etc.



Figura 1.1.: Funcionamiento de los algoritmos RL: El agente inteligente interactúa con el entorno, toma decisiones (acciones) y recibe retroalimentación en forma de recompensas positiva o negativa en función de esas acciones [ZB20]

Durante este proceso, se van descubriendo qué acciones conducen al agente a obtener una recompensa mayor y se va probando con dichas acciones. En los casos mas interesantes, las acciones podrían no solo afectar a la recompensa inmediata sino a todas las subsecuencias de recompensas futuras. Estas dos características de prueba y error, de búsqueda y retraso de recompensa, son las mas importantes a distinguir en el aprendizaje por refuerzo.

El agente de aprendizaje debe de tener conocimiento del estado de su entorno y debe realizar las acciones que afecten a ese estado, siguiendo el esquema de la Figura 1.2. Este esquema del RL puede formalizarse como un proceso de decisión de Markov (MDP) ya

que tiene exactamente sus tres elementos: estados S_t , acciones A_t y recompensas R_t , para cada interacción t del Agente con el entorno, donde $t \in \mathbb{N}$.

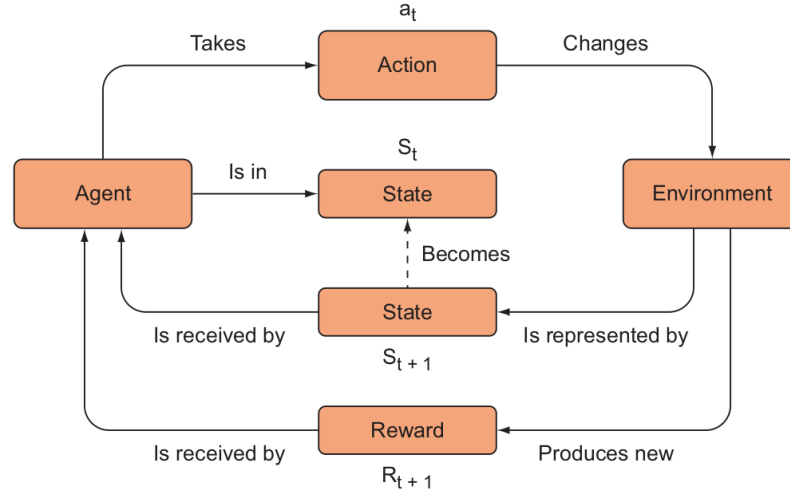


Figura 1.2.: Agente y entorno [ZB20]

El aprendizaje por refuerzo difiere del aprendizaje supervisado ya que nuestro objetivo es actuar correctamente en situaciones que no pueden presentarse en conjuntos de datos de entrenamiento.

A través de las interacciones del agente con el entorno, podemos identificar cuatro elementos principales:

- Una política, que se encarga de definir las interacciones del agente sobre el entorno.
- Una señal de recompensa, que define el objetivo del problema del aprendizaje por refuerzo. En cada paso t , el entorno envía al agente un valor, conocido como señal de recompensa.
- Una función de valor es una función que asigna un valor numérico a cada estado S posible en un entorno de aprendizaje por refuerzo. Este valor representa la cantidad total de recompensa que un agente puede esperar acumular en el futuro a partir de ese estado, esto es, la suma de todas las recompensas obtenidas al ir ejecutando una acción tras otra. Es diferente a la señal de recompensa, que es la recompensa inmediata que el Agente recibe cuando ejecuta una acción en ese estado. La función de valor sería $R = R_1 + R_2 + \dots + R_t + \dots + R_T$ mientras que la recompensa que va obteniendo el agente cada vez que interactúa con el entorno es R_t , $t \in \{1, 2, \dots, T\}$.
- La función acción-valor es una función que asigna un valor numérico a cada par de estado-acción (S, A) en un entorno. Al igual que la función valor, definirá la cantidad total de recompensas que un agente espera acumular en el futuro a partir de ese estado y ejecutando esa acción.

- Un modelo del entorno imita el comportamiento del entorno, ó generalmente, permite hacer inferencias sobre cómo se comportará el entorno. Por ejemplo, dado un estado S_t y una acción A_t , el modelo podría predecir el siguiente estado S'_t resultante y la siguiente recompensa R' .

Los métodos del entorno para resolver problemas de aprendizaje por refuerzo que utilizan modelos y planificación se conocen como *basados en modelo (model-based)*. En cambio los métodos del entorno que no utilizan modelos ni planificación y basan su aprendizaje en prueba y error, se llaman *libres de modelo (model-free)*.

- Un episodio es la secuencia de interacciones que el agente realiza con el entorno. A una única interacción entre el agente y el entorno, se le denomina paso (step).

El aprendizaje por refuerzo depende mucho del concepto de estado, S_t , ya que lo necesita tanto para determinar la política a seguir por el agente como para calcular la función de valor. Informalmente, podemos pensar que el estado es la información que entra y sale del modelo y nos dice cómo es el entorno en un momento de tiempo particular t . Enfocaremos entonces nuestra atención en estudiar el aprendizaje de políticas.

1.2. Métodos para soluciones tabulares

En esta sección, describimos los conceptos y algoritmos de aprendizaje por refuerzo donde tenemos un espacio de estados y acciones simples que pueden ser representados en forma de vectores o tablas. Al ser representados en vectores o tablas finitas, son algoritmos que encuentran exactamente la función de valor óptima y la política óptima. No obstante, solo pueden ser aplicados a problemas pequeños debido a las limitaciones de tamaños en los espacio de estados y acciones.

1.2.1. Métodos acción-valor

El problema del bandido multibrazos (*Multi-armed bandit problem*), también conocido como su símil máquina tragaperras, es un problema particular de la retroalimentación evaluativa no asociativa, y vamos a explorarlo con la intención de presentar varios métodos de aprendizaje básicos.

Este problema de aprendizaje consiste en enfrentarse a una elección de k diferentes acciones, donde después de cada elección se recibe una recompensa numérica. La recompensa numérica sigue una distribución de probabilidad estacionaria que depende de la acción que se seleccionó previamente. El objetivo es maximizar la recompensa total obtenida en un largo tiempo, por ejemplo al haber realizado 1000 selecciones de acciones.

Formalmente, en el tiempo t denotamos a la acción seleccionada como A_t y a la recompensa correspondiente como R_t . La recompensa que se espera obtener cuando se escoge la acción arbitraria a se define como $q_*(a) = \mathbb{E}[R_t | A_t = a]$.

Anteriormente, se ha mencionado que nuestro objetivo es maximizar la recompensa a largo plazo pero tenemos que hablar del posible conflicto que puede haber ya que no es posible explotar y explorar el entorno al mismo tiempo. El conflicto entre exploración y

explotación analiza si es mejor explotar ó explorar. Si decidimos explotar, seleccionamos la acción que tiene mayor recompensa y así explotamos el conocimiento. Sin embargo, si escogemos explorar, exploramos otras opciones distintas a la acción con mayor recompensa.

Existen métodos que admiten un equilibrio tanto en el problema del bandido como en otros problemas. En esta sección veremos algunos métodos donde el equilibrio da resultados mejores que métodos donde simplemente se explote conocimiento.

Función acción-valor

Los métodos de acción-valor se encargan de estimar y representar la función acción-valor que vemos definida a continuación. Estos métodos se encargan de encontrar la mejor política a partir de maximizar la recompensa total esperada a lo largo del tiempo. Cuando estimamos esta función, un agente podría seleccionar las acciones que maximicen el valor esperado para cada estado, para así poder tomar decisiones informadas y adaptativas.

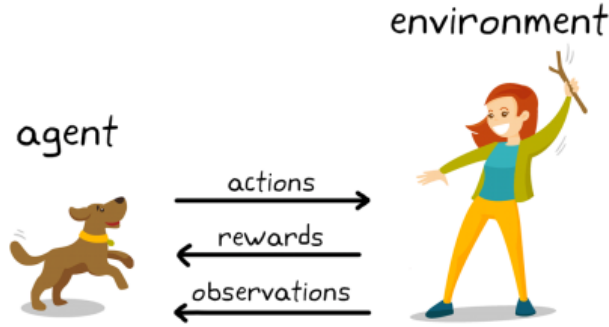


Figura 1.3.: La función acción-valor representa la recompensa que un dueño le da a su perro en función de como de buena sea la acción que realice para un valor de observación del entorno que en este caso es el dueño lanzándole un palo [Mat]

Mas adelante, veremos cómo se promedian las recompensas recibidas con el fin de definir la siguiente **función acción-valor**.

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}[A_i = a]}{\sum_{i=1}^{t-1} \mathbb{1}[A_i = a]} \quad (1.1)$$

- Obsérvese que cuando el denominador tiende a infinito, entonces $Q_t(a)$ converge a $q_*(a)$, $\{Q_t(a)\}_{t \in \mathbb{N}} \rightarrow q_*(a)$. A este método lo llamamos **el promedio de muestras** para estimar los valores de las acciones.
- Este método es apropiado para asignarle valores a las acciones y poder seleccionar la acción greedy, $A_t = \arg \max_a Q_t(a)$. Pero, ¿qué hacemos si tenemos dos acciones

donde sus valores son iguales? Realizaremos una selección entre esas acciones de forma arbitraria.

- Sería interesante que se escogiese una acción aleatoria con cierta probabilidad ϵ y la acción óptima con probabilidad $1 - \epsilon$.
- Así, se obtiene el método **ϵ -Greedy** que explora con probabilidad ϵ y explota con probabilidad $1 - \epsilon$.

$$A_t = \begin{cases} \arg \max_a Q_t(a), & \text{con probabilidad: } 1 - \epsilon \\ a \sim \text{Uniforme}(a_1, a_2, \dots, a_k), & \text{con probabilidad: } \epsilon \end{cases}$$

Como sabemos que nuestro agente no puede explorar y explotar simultáneamente, este método nos da un algoritmo para saber cuándo explotar y que nuestro agente tome la mejor acción actual (la acción que genere la mayor recompensa en el instante t) o cuándo explorar, con el objetivo de mejorar su conocimiento mediante el descubrimiento de acciones que no son las mejores. La exploración de esas acciones y la toma de acciones no codiciosas pueden conducir a obtener las mayores recompensas acumuladas en el futuro.

Implementación incremental de la función acción-valor

Sea R_i la recompensa recibida tras la selección de acción i -ésima, denotamos Q_n como la estimación de su valor acción después de haber seleccionado $n - 1$ acciones, $Q_n = \frac{R_1 + R_2 + \dots + R_{n-1}}{n-1}$. Dado Q_n y la recompensa n -ésima R_n , vamos a ver a continuación como definir Q_{n+1} a partir de estos valores:

$$Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n) \quad (1.2)$$

Demostración.

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i = \frac{1}{n} (R_n + \sum_{i=1}^{n-1} R_i) = \frac{1}{n} (R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i) \\ &= \frac{1}{n} (R_n + (n-1) Q_n) = \frac{1}{n} (R_n + n Q_n - Q_n) = Q_n + \frac{1}{n} (R_n - Q_n) \end{aligned}$$

□

Finalmente, hemos obtenido una regla de actualización incremental al encontrar la definición recursiva del promedio de muestras. Podemos observar como en cada nueva estimación de Q se calcula:

$$\text{NuevaEstimación} \rightarrow \text{ViejaEstimación} + \text{TamañoPaso}[\text{Etiqueta} - \text{ViejaEstimación}]$$

Selección de acciones con un límite de confianza superior

La selección de acciones UCB, conocido por sus siglas en inglés Upper-Confidence Bound, mezcla exploración, con el término $c\sqrt{\frac{\ln t}{N_t(a)}}$, y explotación, con $Q_t(a)$, a través del uso de intervalos de confianza. De esta forma, el agente toma la acción que maximiza la siguiente expresión, donde observamos como hay un término para exploración y otro para la explotación.

$$A_t = \arg \max_a [Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}] \quad (1.3)$$

1.2.2. Procesos de decisión finitos de Markov

Los procesos de decisión finitos de Markov (*Markov Decision Processes, MDP*) formalizan el problema del aprendizaje por refuerzo donde un agente interactúa con un entorno. Los procesos de decisión finitos de Markov forman parte de los procesos de Markov en tiempo discreto. Recordamos a continuación algunas definiciones sobre este campo de estudio.

La propiedad básica que caracteriza a los procesos de Markov es el análogo probabilístico a una propiedad muy frecuente en los sistemas físicos. Si en un sistema se conoce el estado en un determinado instante, éste determina el comportamiento futuro del sistema, independientemente del comportamiento anterior. Esto es lo que se conoce como Principio de independencia entre el pasado y el futuro cuando el presente es conocido.

Formalización Matemática MDPs

Definición 1.1. Una secuencia de σ -álgebra en Ω se dice que es una **filtración de σ -álgebras** si cumple

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \dots \subset \mathcal{F}$$

Definición 1.2. Un **proceso estocástico** es una colección de variables aleatorias $\{X_t\}_{t \in T}$, con $T \subseteq \mathbb{R}$. Si T es un conjunto numerable, entonces el proceso estocástico se dice que es de **tiempo discreto**.

Si $\{X_n\}_{n \in \mathbb{N}}$ es un proceso estocástico en tiempo discreto definido sobre un espacio probabilístico $(\Omega, \mathcal{A}, P)$ y las variables del proceso X_n no toman valores en todo $\mathbb{R}$, sino en un subconjunto $E \subset \mathbb{R}$, entonces este proceso se denomina cadena y su espacio de estados es $(E, \mathcal{B}_E)$.

Definición 1.3. Sea $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ una filtración de σ -álgebras, esto es, una sucesión no decreciente de sub- σ -álgebras de $\mathcal{A}$. Se dice que $\{X_n\}_{n \in \mathbb{N}}$ es un **proceso estocástico adaptado a la filtración** si X_n es $\mathcal{F}_n$ -medible, $\forall n \geq 0$.

Definición 1.4. Un proceso estocástico en tiempo discreto, $\{X_n\}_{n \in \mathbb{N}}$, definido sobre $(\Omega, \mathcal{A}, P)$ y con espacio de estados $(E, \mathcal{B}_E)$ es un **proceso de Markov respecto de la filtración** $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$, si solamente si,

1. $\forall n \geq 0, X_n$ es $\mathcal{F}_n$ – medible (adaptado a la filtración)
2. $\forall n \geq 1, \forall B \in \mathcal{B}_E, P[X_n \in B / \mathcal{F}_{n-1}] = P[X_n \in B / X_{n-1}]$ c.s.

Se dice que un proceso estocástico es un proceso de Markov, si es de Markov respecto a la filtración natural asociada al proceso. Así hablaremos simplemente de procesos de Markov, que es la formalización matemática del Aprendizaje por Refuerzo.

Definición 1.5. Sea $\{X_n\}_{n \in \mathbb{N}}$ un proceso estocástico en tiempo discreto definido sobre el espacio probabilístico $(\Omega, \mathcal{A}, P)$ y con espacio de estados $(E, \mathcal{B}_E)$, $E \subset \mathbb{R}$. Se dice que $\{X_n\}_{n \in \mathbb{N}}$ es un **proceso de Markov**, si y solamente si,

$$\forall n \geq 1, \forall B \in \mathcal{B}_E, P[X_n \in B / X_0, \dots, X_{n-1}] = P[X_n \in B / X_{n-1}], \text{ c.s.}$$

Tras esta formulación matemática, nos habremos dado cuenta como el agente y el entorno interactúan en pasos (steps) de tiempos discretos, de aquí en adelante lo notaremos $t \in T$. En cada paso de tiempo t , el agente observa el estado actual S_t y en función de ese estado selecciona una acción (una elección entre todas las posibles acciones para ese estado, $A_t \in \mathcal{A}(S_t)$). A continuación, el entorno transita a un nuevo estado, S_{t+1} , y emite una recompensa, R_{t+1} .

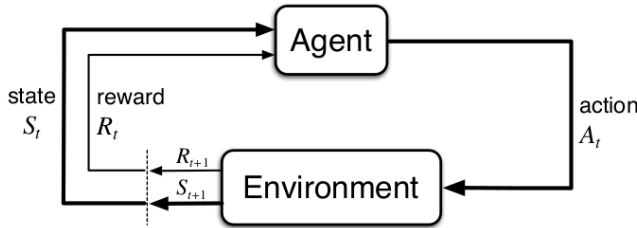


Figura 1.4.: Esquema del aprendizaje por refuerzo. Este paradigma está basado en ejecutar la siguiente acción a partir del estado del entorno actual y obtener la siguiente recompensa junto con el siguiente estado [SB18]

El MDP y el agente juntos nos dan una secuencia de trayectorias que comienzan de la forma: $S_0, A_0, S_1, A_1, R_2, S_2, A_3, R_3, \dots$

En un MDP finito, los conjunto de estados, acciones y recompensas tienen todos un número finito de elementos. Para las variables aleatorias R_t y S_t definimos una distribución de probabilidad condicionada al estado anterior y a la acción realizada en el paso t , $p(s', r | s, a) = Pr\{S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a\}$, $\forall s, s' \in \mathcal{S}, a \in \mathcal{A}(s), r \in \mathcal{R}$

Esta formulación permite modelar muchos problemas del mundo real ya que para aplicar aprendizaje por refuerzo bastará con formular el problema como un MDP.

Políticas y funciones de valor

Las funciones de estados, s , o de la pareja estado-acción, (s, a) , estiman *cómo de bueno* es un agente para un estado o para una acción seleccionada dado un estado. Para

determinar *cómo de bueno* es un agente se basan en las recompensas que obtienen tras ejecutar cada acción, esto es, estudian la recompensa futura que espera recibir el agente.

Procedemos a definir los conceptos de política, función de estado y función de la pareja estado-acción.

- Una **política** π establece un mapeo entre el espacio de estados $\mathcal{S}$ y el espacio de acciones $\mathcal{A}$, $\mathcal{S} \rightarrow \mathcal{A}$. La política es la encargada de asignar una probabilidad a cada acción para cada posible estado del entorno. Esto es, si el agente sigue la política π en el paso t entonces $\pi(a|s)$ es la probabilidad de que $A_t = a$ cuando $S_t = s$, $\pi(\cdot|s) : \mathcal{A} \rightarrow [0, 1]$
- La **función valor** de un estado s bajo una política π , denotada como $v_\pi(s)$, es el valor esperado de la recompensa cuando comenzamos con un estado s siguiendo una política π . Para los MDPs, lo definimos de la siguiente forma:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- La **función acción-valor** para un estado s bajo una política π , denotada como $q_\pi(s, a)$, es valor esperado de la recompensa cuando comenzamos con un estado s , realizando la acción a , siguiendo una política π . Para los MDPs, cambiamos la definición que realizamos en la Ecuación 1.1 y la definimos de la siguiente forma:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Las funciones de valor definen un orden en las políticas:

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \quad \forall s \in \mathcal{S}$$

La política óptima se define como la política que es mayor o igual al resto de políticas, aunque podría ser mas de una, por eso la denotaremos π_* . Hay que tener en cuenta que podemos definir mas de una política óptima pero siempre existe al menos una.

- $v_*(s) = \max v_\pi(s) \quad \forall s \in \mathcal{S}$
- $q_*(s, a) = \max q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

Ecuaciones de Bellman

Son ecuaciones que expresan una relación entre el valor de un estado y los valores de los estados sucesores teniendo en cuenta las recompensas y las probabilidades de transición en un proceso de decisión estocástico.

Se definen desarrollando la definición de esperanza matemática y aplicando, en la definición de función de valor y acción-valor, la expresión de la recompensa acumulada $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, $\gamma \in [0, 1]$. Después, promediamos todos los estados posibles desde el estado actual, ponderando cada probabilidad de ocurrencia de un estado, como podemos ver a continuación.

- $v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_\pi(s')], \quad \forall s \in \mathcal{S}$
- $q_\pi(s, a) = \sum_{s', r} p(s', r|s, a) [r + \gamma q_\pi(s', a)], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$

Programación dinámica

La programación dinámica (*Dynamic Programming, DP*) se refiere a la colección de algoritmos que pueden ser usados para el cálculo de políticas óptimas dado un perfecto modelo del entorno como un MDP. Su idea clave, y también del aprendizaje por refuerzo, es el uso de funciones de valor para organizar y estructurar la búsqueda de buenas políticas. Estas políticas óptimas van a satisfacer las funciones de valor óptimas, v_* y q_* , que son conocidas como las **ecuaciones de optimalidad de Bellman**.

- $v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$
- $q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')]$

La programación dinámica nos da un algoritmo conocido como **iteración de políticas** que se basa en dos pasos:

1. Evaluación de política: calcular la función de valor v_{π_0} para una política arbitraria inicial π_0 . Para producir cada sucesiva aproximación de $v_{\pi_{k+1}}$ de v_{π_k} , se va iterando la política aplicando la misma operación en cada estado s . En cada iteración se actualiza el valor en cada estado para producir $v_{\pi_{k+1}}$ que se utilizará en la siguiente aproximación.
2. Mejora de política: el objetivo del calculo iterativo de sucesivas aproximaciones de v_{π_k} tiene como objetivo ayudar a encontrar la mejor política. Este proceso compara las políticas y selecciona como política π_k a la mejor políticas de las anteriores de la secuencia.

Inicialmente, tenemos la política π que se mejora usando la función v_π y así se obtiene la política π' . Así se va obteniendo una secuencia de políticas mejores y de funciones de valores:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Por otra parte, tenemos otro algoritmo conocido como **iteración de valor** donde se combina la evaluación de política con la mejora de política logrando una convergencia mas rápida. La iteración de valor obtiene mejores resultados de convergencia porque es un algoritmo que actúa igual que iteración de políticas pero intercala múltiples pasadas de evaluación de políticas entre cada pasada de mejora de política.

1.2.3. Métodos de Monte Carlo

Los métodos de Monte Carlo son los primeros métodos de aprendizaje que vamos a estudiar cuyo objetivo es estimar v_π , $v \approx v_\pi$. Estos métodos no necesitan un conocimiento del entorno, aprenden directamente a través de su interacción. Solo necesitan experiencias, esto es, muestras de secuencias de estados, acciones y recompensas obtenidas a través de interacciones del agente con el entorno.

Las experiencias se obtienen en cada episodio ya que cada episodio contiene un número n de pasos, $t \in T$ donde $T = \{t_1, \dots, t_n\}$. Cuando el episodio finaliza, se estima v_π y la política cambia. Es importante resaltar que la política y las aproximaciones de la función de valor, $v \approx v_\pi$, se realizan episodio a episodio y no paso a paso.

Si nos damos cuenta, al tener múltiples episodios es como si tuviésemos un problema del bandido por cada episodio del entorno donde los diferentes problemas del bandido estuviesen interrelacionados.

Predicción con Monte Carlo

Son métodos que nos permiten estimar la función de valor, $v_* \approx v_\pi$, de un estado individual independiente de los valores de cualquier otro estado. Anteriormente, en programación dinámica, el valor de cada estado dependía de los valores anteriores de los estados de la secuencia, ahora el cálculo de la estimación es independiente. Además, no depende del tamaño del MDP (número total de pasos considerando todos los episodios), solo depende de la duración de cada episodio.

Control con Monte Carlo

Son métodos que se encargan de estimar la función acción-valor, $q_* \approx q_\pi$, usando Monte Carlo y entendiendo la importancia de mantener la exploración.

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} q_*$$

De acuerdo a la idea de generalización de iteración de políticas, la política óptima se obtiene de la anterior secuencia donde vamos alternando un paso de evaluación de política con uno de mejora de política.

Utilizan un algoritmo llamado **Monte Carlo ES (Exploring Starts)** para estimar $\pi \sim \pi^*$, donde se consigue que las recompensas se acumulen y se promedien, sin importar la política que se estuviese utilizando. Este algoritmo no converge a ninguna política subóptima.

Tenemos que tener en cuenta que existen métodos de control *on-policy*, donde se intentan evaluar y mejorar la política que se esta usando para tomar decisiones, y métodos de control *off-policy*, que intentan evaluar y mejorar una política diferente a la que se está usando en la toma de decisiones del agente en su entorno.

1.2.4. Aprendizaje de diferencia temporal

El aprendizaje de diferencia temporal presenta algunas singularidades con respecto a los aprendizajes anteriores. Con respecto a Monte Carlo, el aprendizaje de diferencia temporal puede actualizar los valores en cada paso mientras que Monte Carlo solo actualizaba su función de valor al final de cada episodio. Este aprendizaje converge asintóticamente a las predicciones correctas y convergen más rápidamente que los métodos de Monte Carlo presentados anteriormente. A continuación, estudiaremos como actualizar la función de valor para que realice modificaciones en cada paso de cada episodio.

Predicción TD

El método mas simple de diferencia temporal realiza la siguiente actualización:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

donde en el paso $t + 1$ se realiza la actualización, utilizando la recompensa R_{t+1} , la estimación de la función de valor $V(S_{t+1})$ y como etiqueta estamos utilizando el valor $R_{t+1} + \gamma V(S_{t+1})$ mientras que Monte Carlo utilizaba G_t debido a que no podía actualizar paso a paso y por tanto, consideraba la recompensa acumulada de todo el episodio como objetivo.

Un simple método de Monte Carlo, first-visit (adecuado para entornos no estacionarios), actualizaría la función valor de la siguiente forma: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$.

Control On-Policy TD: SARSA

SARSA aplica el patrón de iteración de política generalizada y lo combina con el aprendizaje TD con objeto de encontrar las mejores políticas.

Su primer paso es aprender una función de acción-valor. Va a ir considerando transiciones de parejas de estados-acciones a parejas de estados-acciones y aprendiendo de sus valores. Los teoremas aseguran la convergencia en TD(0) aplicando:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Cuando llega a un estado terminal, entonces define $Q(S_t, A_t) = 0$. El nombre del algoritmo, **SARSA**, viene de la quintupla necesaria para una actualización $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

Control Off-Policy TD: Q-Learning

Uno de los primeros avances en el aprendizaje por refuerzo fue el desarrollo de un algoritmo de control TD *off-policy* conocido como **Q-learning** [WD92] y definido por:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Este algoritmo difiere de SARSA porque aprendemos la función acción-valor, aproxima directamente la función acción-valor óptima q_* independientemente de la política que este siguiendo, intentamos obtener una política diferente a la que se esta usando y la vamos mejorando (*off-policy*). Esto se consigue gracias a que utiliza las ecuaciones de optimalidad de Bellman (Sección 1.2.2), se toma la acción que produce mayor recompensa de media al realizar $\max_a Q(S_{t+1}, a)$, mientras que antes utilizábamos las ecuaciones de Bellman (Sección 1.2.2) para aproximar una función acción-valor pero no se consideraba óptima.

- SARSA $\sim \pi$
- Q-learning $\sim \pi_* \neq \pi$

Expected SARSA

Es un algoritmo que calcula la esperanza de Q bajo su política en lugar de maximizar Q sobre la pareja estado-acción. Por tanto, tenemos

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)] \\ &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)] \end{aligned}$$

Es computacionalmente más complejo que SARSA pero a cambio de ello admite una menor varianza a la hora de seleccionar aleatoriamente una acción de A_{t+1} . Asimismo, **Expected SARSA** engloba y generaliza Q-learning. De hecho, sus resultados suelen mejorar a otros TD más conocidos, con el inconveniente de un mayor coste computacional.

1.2.5. n-steps Bootstrapping

Predicción: n-step TD

Nos cuestionamos la existencia de un conjunto de métodos intermedios que se encuentren entre los métodos de Monte Carlo y los métodos de TD. Los métodos de Monte Carlo realizan una actualización de la función basándose en la secuencia completa de recompensas observadas desde el estado inicial hasta el estado final del episodio. Mientras que, la actualización de los métodos TD se basa únicamente en la siguiente recompensa: si estamos en un estado S_t , solo nos interesaría la recompensa R_{t+1} . Los métodos de Monte Carlo necesitarían las recompensas, $R_{t+1}, \dots, R_s$, siendo el paso s donde se alcanza el estado final S_s .

Un método intermedio sería un algoritmo que necesitase n recompensas, $R_{t+1}, \dots, R_{t+n}$, sin imponer que el estado obtenido al alcanzar la recompensa en el step $t+n$ tenga que ser un estado final. Estos métodos son también métodos TD y se conocen como **n-steps TD**. Un algoritmo de aprendizaje de estado-valor n -step TD realizaría lo siguiente en un episodio que contenga un número de pasos de T .

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)]$$

Control: n-step SARSA

El método n -steps TD puede utilizarse como método de predicción pero no como método de control. Como método de control surge un método n -steps que se combina con SARSA para producir un método *on-policy* de control TD. Este algoritmo se conoce como **n-step SARSA** y se define de la forma:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

1.2.6. Resumen

Todos los métodos que hemos explorado tienen tres ideas claves que comparten:

1. Buscan estimar funciones de valor: v_π y q_π
2. Para realizar su estimación, calculan sus valores a lo largo de las trayectorias de estados posibles.
3. Siguen la estrategia de iteración de políticas generalizadas(*Generalized Policy Iteration*, *GPI*): mantienen una función de valor aproximada y una política aproximada, y continúan tratando de mejorar cada una basándose en la otra.

1.3. Métodos para soluciones aproximadas

En la segunda parte, tratamos de extender los métodos tabulares presentados en la primera parte para aplicarlos a problemas con espacios de estados arbitrariamente grandes ya que los métodos anteriores tenían la ventaja de ser métodos exactos pero la desventaja de que solo se podían aplicar a espacios pequeños. Sin embargo, en muchas tareas del aprendizaje por refuerzo, el espacio de estados es combinatorio y enorme. En esos casos, es imposible encontrar una política óptima, π_* , o una función de valor óptima, v_{π_*} . Trataremos de encontrar una solución aproximada usando recursos computacionales limitados.

El problema con los espacios de estados grandes no es solo la memoria necesaria para tablas grandes, sino también el tiempo y los datos necesarios para ir actualizando las tablas.

Tenemos que tener en cuenta que la **generalización** es importante si queremos encontrar una solución aproximada en un espacio de datos combinatorio y enorme, de hecho, es ideal tener generalización y discriminación en cualquier tipo de aprendizaje. El tipo de generalización que necesitamos se denomina **aproximación de funciones**, ya que toma ejemplos de una función deseada e intenta generalizar a partir de ellos para construir una aproximación de la función completa. La aproximación de funciones es un ejemplo de aprendizaje supervisado que aplicaremos al aprendizaje por refuerzo, teniendo en cuenta que aparecerán una serie de problemas que en el aprendizaje supervisado no estaban presentes.

Si aplicamos redes neuronales profundas al aprendizaje por refuerzo, entonces estamos aplicando aproximación de funciones para obtener Aprendizaje “Profundo” por Refuerzo (*Deep Reinforcement Learning*, *DRL*).

A continuación, veremos distintos enfoques, comenzando por el caso de la predicción donde se proporciona la política y solo se aproxima su función de valor, siguiendo por los casos de control donde se aproxima la función acción-valor con objeto de alcanzar una política óptima. También, trataremos la aproximación de políticas con métodos *off-policy* y los métodos que aproximan políticas a partir de los gradientes sin necesidad de aproximar una función de valor. Finalmente, veremos una lista de algoritmos que pertenecen al aprendizaje profundo por refuerzo (DRL).

1.3.1. Parametrización de la función valor

La función valor parametrizada se define como $v(s, \omega) \approx v_\pi$, donde $\omega \in \mathbb{R}^d$ es el vector de pesos. Por ejemplo, la aproximación por funciones lineales se puede notar como $v(s, \omega) = \sum_i \omega_i x_i(s)$, siendo $x(s) = (x_1(s), x_2(s), \dots, x_d(s))$ el vector de características del estado s y tiene la misma longitud que ω . Otro ejemplo de aproximación, sería la función calculada por una red neuronal definida como $v(s, \omega)$ donde ω es el vector de pesos de conexión entre todas las capas de la red.

Si $\{(s_1, v_\pi(s_1)), (s_2, v_\pi(s_2)), \dots\}$ es un conjunto de entrenamiento, entonces nuestro objetivo es obtener los pesos w que definen la función $v(s, \omega)$ que aproxima a la función $v_\pi(s)$. Para ello, necesitaremos una medida que calcule el error que existe entre las dos funciones y utilizaremos la función del error cuadrático medio de la función de valor definida como

$$VE = \sum_{s \in \mathcal{S}} \mu(s) [v_\pi(s) - v(s, \omega)]^2$$

donde $\mu(s)$ es una distribución de probabilidad de estados, verificando $\mu(s) \geq 0$ y $\sum_s \mu(s) = 1$.

La evaluación de políticas bajo funciones de aproximación requiere especificar un función objetivo a minimizar y el error VE es un objetivo posible. Centraremos nuestra atención en los métodos de aproximación de funciones basados en principios de gradiente, y en particular en los métodos de descenso de gradiente lineales. Nos enfocamos en estos métodos en parte porque los consideramos particularmente prometedores y porque revelan problemas teóricos clave.

Métodos del gradiente estocástico

Los métodos del gradiente estocástico (SGD) son ampliamente utilizados para aproximar funciones y son muy adecuados para el aprendizaje por refuerzo. El vector de pesos es un vector columna con un número fijo de componentes, $\omega = (\omega_1, \omega_2, \dots, \omega_d)^T$, y la función de valor aproximada es $v(s, \omega)$ es una función diferenciable de ω para todos los estados $s \in \mathcal{S}$. Actualizamos ω en cada tiempo de paso discreto $t = 0, 1, 2, 3, \dots$, notamos w_t los pesos en cada paso de tiempo, entonces tenemos la aplicación $S_t \rightarrow v_\pi(S_t)$.

Los métodos del gradiente descendente estocástico (SGD) actualizan los pesos después de cada paso (step) de muestras de la siguiente forma, donde α indica el tamaño de paso (step):

$$\begin{aligned} \omega_{t+1} &= \omega_t - \frac{1}{2} \alpha \nabla [v_\pi(S_t) - v(S_t, \omega_t)]^2 \\ &= \omega_t + \alpha [v_\pi(S_t) - v(S_t, \omega_t)] \nabla v(S_t, \omega_t) \end{aligned}$$

No buscamos una aproximación que tenga un error VE cero en todos los estados, más bien buscamos una aproximación que equilibre los errores en diferentes estados.

Procedemos a estudiar la aplicación $S_T \rightarrow U_t$ donde U_t es una versión corrupta-ruidosa de $v_\pi(S_t)$ y tendríamos

$$\omega_{t+1} = \omega_t + \alpha [U_t - v(S_t, \omega_t)] \nabla v(S_t, \omega_t)$$

Si U_t es una estimación imparcial, esto es, si $\mathbb{E}[U_t|S_t = s] = v_\pi(S_t)$ para cada t , entonces hay garantía de que w_t converge a un óptimo local bajo las condiciones de aproximación estocásticas de ir decrementando α . Si tomamos $U_t = G_t$, tenemos el algoritmo del **gradiente de Monte Carlo**.

Métodos de semi-gradientes

Una razón por la que se suele permitir este enfoque es porque significa un aprendizaje más rápido, continuo y en línea, sin tener que esperar el final de un episodio. Esto les permite ser utilizados en problemas continuos y proporciona ventajas computacionales. Un método de semigradiiente prototípico es el semigradiiente TD(0).

El **calculo del gradiente en el algoritmo TD** se dice que es semi-gradiente ya que $\nabla U_t \neq 0$ y se actualizarían de la siguiente forma donde $U_t = R_{t+1} + \gamma v(S', \omega_t)$

$$w_{t+1} \rightarrow w_t + \alpha[U_t - v(S_t, \omega_t)]\nabla v(S_t, \omega_t).$$

El gradiente de Monte Carlo con agregación de estados

La agregación de estados es una forma simple de generalizar las aproximaciones de funciones en las que los estados se agrupan juntos, con un valor estimado (un componente del vector de pesos ω) para cada grupo. Por ejemplo, si tenemos 1000 estados, los podemos agrupar de 100 en 100. Para calcular el gradiente de Monte Carlo con agregación de estados realizaremos las siguientes actualizaciones para cada paso $t \in \{1, 2, \dots, T\}$

$$\omega_{t+1} = \omega_t + \alpha[G_t - v(S_t, \omega_t)]\nabla v(S_t, \omega_t)$$

Si comparamos el gradiente de Monte Carlo con agregación de estados y el gradiente de TD, nos damos cuenta que TD converge más rápido que el gradiente de Monte Carlo. Es más, TD converge al sesgo del valor estimado.

Métodos lineales

Vamos a estudiar la actualización lineal de TD ya que uno de los casos mas importantes en aproximaciones de funciones es cuando $v(s, \omega)$ es una función lineal del vector de pesos ω . Así tenemos $v(s, \omega) = \omega^T x(s) = \sum_{i=1}^d \omega_i x_i(s)$ donde el vector $x(s)$ se conoce como vector de características representado para el estado s y $\nabla v(s, \omega) = x(s)$. Entonces la actualización de SGD se reduce a:

$$\omega_{t+1} = \omega_t + \alpha[U_t - v(S_t, \omega_t)]x(S_t)$$

Los métodos lineales tienen grandes ventajas ya que son simples de entender y analizar matemáticamente. Con buenas características, los métodos lineales pueden aprender rápidamente y lograr buenos valores de precisión, notando $x_t = x(S_t)$ obsérvese como

queda nuestro objetivo de actualización.

$$\begin{aligned}
\omega_{t+1} &= \omega_t + \alpha[U_t - v(S_t, \omega_t)]x(S_t) \\
&= \omega_t + \alpha[R_{t+1} + \gamma v(S', \omega_t) - v(S_t, \omega_t)]x(S_t) \\
&= \omega_t + \alpha[R_{t+1} + \gamma \omega_t^T x_{t+1} - \omega_t^T x_t]x_t \\
&= \omega_t + \alpha[R_{t+1}x_t - x_t(x_t - \gamma x_{t+1})^T \omega_t]
\end{aligned}$$

Una vez que alcance un estado estable, para un ω_t , la esperanza del próximo vector de pesos se podrá escribir como:

$$\mathbb{E}[\omega_{t+1}|\omega_t] = \omega_t + \alpha(b - A\omega_t)$$

donde $b = \mathbb{E}[R_{t+1}x_t] \in \mathbb{R}^d$ y $A = \mathbb{E}[x_t(x_t - \gamma x_{t+1})] \in \mathbb{R}^d x \mathbb{R}^d$.

Si el sistema lineal converge, debe converger al vector de pesos ω_{TD} :

$$b - A\omega_{TD} = 0 \rightarrow b = A\omega_{TD} \rightarrow \omega_{TD} = A^{-1}b$$

Este vector se conoce como punto fijo TD y de hecho, el semi gradiente TD lineal converge a este punto fijo. Existe la siguiente relación donde el valor VE del punto fijo se encuentra acotado:

$$VE(\omega_{TD}) \leq \frac{1}{1-\gamma} \min_{\omega} VE(\omega)$$

1.3.2. Parametrización de la función acción-valor

Volvemos al problema del control con una aproximación paramétrica de la función acción-valor $q(s, a, \omega) \approx q^*(s, a)$ donde $\omega \in \mathbb{R}^d$ es un vector de pesos de dimensión finita. Además, sabemos que $q(s, a, \omega)$ se define como $q(s, a, \omega) = \omega^T x(s, a)$.

Por ejemplo, si $x(s) = [x_0(s), x_1(s), x_2(s), x_3(s)]$, $\mathcal{A}(s) = \{a_0, a_1, a_2\}$, entonces $x(s, a) = [x_0(s), x_1(s), x_2(s), x_3(s), x_0(s), x_1(s), x_2(s), x_3(s), x_0(s), x_1(s), x_2(s), x_3(s)]$

Episodic SARSA con aproximación de funciones

La extensión de los métodos de predicción para los métodos de control es directa, $q \sim q_{\pi}$. La actualización de U_t puede ser cualquier aproximación de $q_{\pi}(S_t, A_t)$ con una red neuronal.

El método *episodic SARSA semi-gradiente one-step* realiza la siguiente actualización de pesos en su red neuronal con objeto de definir una función $q(S_t, A_t, \omega)$ que aproxime a $q_{\pi}(S_t, A_t)$, $q(S_t, A_t, \omega) \approx q_{\pi}(S_t, A_t)$:

$$\omega_{t+1} \rightarrow \omega_t + \alpha \delta \nabla Q(s, a; \omega)$$

donde $\delta = Q(s, a) - Q(s, a; \omega)$ es el error de la estimación.

Expected SARSA con aproximación de funciones

Recordemos la formulación de actualización de la función acción-valor SARSA vs Expected SARSA en los métodos tabulares 1.2.4, ahora modificamos estas actualizaciones como vemos a continuación. Si aplicamos aproximaciones de funciones para aproximar la función q , entonces tendríamos una función $q(s, a, \omega)$ donde los pesos se irían actualizando de la siguiente forma:

- SARSA:

$$\omega_{t+1} \rightarrow \omega_t + \alpha(R_{t+1} + \gamma q(S_{t+1}, A_{t+1}, \omega_t) - q(S_t, A_t, \omega_t)) \nabla q(S_t, A_t, \omega_t)$$

- Expected SARSA:

$$\omega_{t+1} \rightarrow \omega_t + \alpha(R_{t+1} + \gamma \sum \pi(a'|S_{t+1}) q(S_{t+1}, a', \omega_t) - q(S_t, A_t, \omega_t)) \nabla q(S_t, A_t, \omega_t)$$

Obsérvese como Q-learning es un caso especial de Expected SARSA donde se tiene:

$$\omega_{t+1} \rightarrow \omega_t + \alpha(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a', \omega_t) - q(S_t, A_t, \omega_t)) \nabla q(S_t, A_t, \omega_t)$$

La recompensa promedio

La recompensa promedio es una nueva formulación de los problemas de control y se define como:

$$r(\pi) = \lim_{h \rightarrow \infty} \frac{1}{h} \sum_{t=1}^h \mathbb{E}[R_t | S_0, A_{0:t-1} \sim \pi] = \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) r$$

Al haber definido la recompensa promedio, entonces nuestro objetivo cambia de $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots$ a definirse en función de diferencias entre la recompensa y la recompensa promedio $r(\pi)$, $G_t = R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \dots$ y se conoce como retorno diferencial.

Los retornos diferenciales representan la cantidad de recompensa superior a la recompensa media que recibirá el Agente. Del mismo modo que hemos definido la recompensa promedio, podemos definir la función de valor como:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \sum_{s', r} p(s', r | s, a) (r - r(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a'))$$

Muchos algoritmos pueden reescribirse y utilizar la recompensa promedio, un ejemplo de ello es SARSA con semigradiiente diferencial para estimación $q \approx q^*$.

1.3.3. Métodos de gradientes de políticas

Anteriormente, hemos estado aprendiendo las funciones de valor y ahora enfocamos nuestro estudio en aprender políticas directamente. Para ello, parametrizaremos nuestra política, $\pi(a|s)$, y aprenderemos los pesos θ de la función $\pi(a|s, \theta)$, $\pi(a|s, \theta) \approx \pi(a|s)$.

Esta función se define como la probabilidad de la acción a en el tiempo t dado el estado s del entorno y el parámetro θ del tiempo t , esto es, $\pi(a|s, \theta) = \Pr\{A_t = a | S_t = s, \theta_t = \theta\}$ y $\pi(a|s, \theta) \approx \pi(a|s)$.

En esta sección veremos métodos basados en el cálculo del gradiente para aprender los parámetros θ de la política parametrizada $\pi(a|s, \theta)$. Todos los métodos seguirán el esquema general conocido como métodos de gradientes de políticas.

Aprendiendo políticas parametrizadas

Denotamos nuestro vector de parámetros de políticas como θ y nuestra política parametrizada como $\pi(a|s, \theta)$, nos indica la probabilidad de realizar la acción a estando el agente en el estado s y los parámetros θ nos marcan cuanto se aproxima la función a $\pi(a|s)$. La política parametrizada cumple las siguientes restricciones:

1. $\pi(a|s, \theta) \geq 0 \quad \forall a \in \mathcal{A} \text{ y } s \in \mathcal{S}$
2. $\sum_{a \in \mathcal{A}} \pi(a|s, \theta) = 1 \quad \forall s \in \mathcal{S}$

Para satisfacer estas condiciones con una función parametrizada, no podemos usar una función lineal directamente como hicimos con la función de valor. Necesitamos la función softmax y usar la parametrización de políticas softmax:

$$\pi(a|s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_{b \in \mathcal{A}} e^{h(s,b,\theta)}}$$

Con esta función garantizamos que la política sea positiva al usar una función exponencial y al dividir entre el denominador tenemos un valor normalizado. Nos hemos trasladado los valores de la función h a valores en el intervalo $[0, 1]$.

Gradiente de políticas

Nuestra finalidad es maximizar la recompensa a largo plazo $R_t, R_{t+1}, R_{t+2}, \dots$,

Formular un objetivo para aprender una política parametrizada en cierto sentido es más sencillo de lo que era para los métodos de acción basados en valores. El objetivo final del aprendizaje por refuerzo es aprender una política que obtenga la mayor recompensa posible a largo plazo y resulta que al parametrizar la política lo estamos teniendo directamente como objetivo de aprendizaje.

En tareas episódicas, $G_t = \sum_{t=0}^T R_t$ y en tareas continuas, $G_t = \sum_{t=0}^{\infty} \gamma^t R_t$ o $G_t = \sum_{t=0}^{\infty} \gamma^t R_t - r(\pi)$. Por tanto, podemos usar la recompensa promedio para la optimización de políticas.

A continuación describimos el **teorema del gradiente de políticas** que será un resultado clave que nos permite escribir el gradiente de la recompensa promedio para que sea más fácil estimar a partir de la experiencia. Este teorema usará métodos basados en SGD para obtener el gradiente del objetivo de la siguiente forma:

$$\begin{aligned} \nabla r(\pi) &= \nabla \sum_s \mu(s) \sum_a \pi(a|s, \theta) \sum_{s', r|s, a} r \\ &= \sum_s \nabla \mu(s) \sum_a \pi(a|s, \theta) \sum_{s', r|s, a} r + \sum_s \mu(s) \nabla \sum_a \pi(a|s, \theta) \sum_{s', r|s, a} r \end{aligned}$$

Teorema 1.1. Teorema del gradiente de política:

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a|s, \theta) q_{\pi}(s, a) \quad (1.4)$$

Actor-Crítico para tareas continuas

Queremos impulsar un algoritmo de descenso de gradiente para nuestra política ya que el teorema anterior nos proporciona nuestro objetivo y su gradiente. Entonces, tenemos que averiguar cómo aproximar este gradiente. Con la siguiente actualización hay garantía de que llegaremos a un punto estacionario:

$$\theta_{t+1} \rightarrow \theta_t + \alpha \sum_a \nabla \pi(a|S_t, \theta_t) q_\pi(S_t, a)$$

Si examinamos todo esto desde el punto de vista de la esperanza matemática, obteniendo muestras estocásticas con una acción y nos damos cuenta de que:

$$\sum_a \nabla \pi(a|S, \theta) q_\pi(S, a) = \sum_a \pi(a|S, \theta) \frac{1}{\pi(a|S, \theta)} \nabla \pi(a|S, \theta) q_\pi(S, a) = \mathbb{E}_\pi \left[\frac{\nabla \pi(A|S, \theta)}{\pi(A|S, \theta)} q_\pi(S, A) \right]$$

Por tanto, la nueva actualización del gradiente estocástico se definiría como:

$$\theta_{t+1} = \theta_t + \alpha \frac{\nabla \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} q_\pi(S_t, A_t)$$

y aplicando la regla de la derivada de un logaritmo, $\nabla \ln \pi(A_t|S_t, \theta_t) = \frac{\nabla \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)}$, se tiene:

$$\theta_{t+1} = \theta_t + \alpha \nabla \ln \pi(A_t|S_t, \theta_t) q_\pi(S_t, A_t)$$

Finalmente, tenemos una **regla de aprendizaje de gradiente de políticas** que podemos aplicar a un algoritmo de recompensa promedio. Procedemos a ver cómo podemos aplicarlos al cálculo del algoritmo de recompensa promedio de semi-gradiente TD(0).

Tendríamos la siguiente aproximación de funciones:

$$\theta_{t+1} = \theta_t + \alpha \nabla \ln \pi(A_t|S_t, \theta_t) [R_{t+1} - \bar{R} + v(S_{t+1}, \omega)] \quad (1.5)$$

donde restar $v(S_t, \omega)$ ya que esto no afecta a la actualización de θ_{t+1} y así conseguimos el error TD, lo demostramos a continuación tomando la esperanza del error TD:

Demostración.

$$\begin{aligned} & \mathbb{E}_\pi [\nabla \ln \pi(A_t|S_t, \theta_t) [R_{t+1} - \bar{R} + v(S_{t+1}, \omega) - v(S_t, \omega)] | S_t = s] \\ &= \mathbb{E}_\pi [\nabla \ln \pi(A_t|S_t, \theta_t) [R_{t+1} - \bar{R} + v(S_{t+1}, \omega)] | S_t = s] - \mathbb{E}_\pi [\nabla \ln \pi(A_t|S_t, \theta_t) v(S_t, \omega) | S_t = s] \\ &= \mathbb{E}_\pi [\nabla \ln \pi(A_t|S_t, \theta_t) [R_{t+1} - \bar{R} + v(S_{t+1}, \omega)] | S_t = s] - 0 \end{aligned}$$

□

Existe un algoritmo llamado Actor-Crítico que implementa esta idea, con un crítico para aprender la función valor para el actor. Para acciones continuas podemos utilizar la política softmax con el algoritmo actor-crítico.

Algorithm 1: Actor-Critico para estimar $\pi_\theta \approx \pi_*$

Entrada: $\pi(a|s, \theta)$ Salida: $v(s, \omega)$

1. Inicializar $\bar{R} \in \mathbb{R}$ a 0
 2. Inicializar los pesos $\omega \in \mathbb{R}^d$ y los parámetros de la política $\theta \in \mathbb{R}^d$
 3. Para cada step hacer:
 - $A \sim \pi(\cdot|S, \theta)$
 - A partir de la acción A , obtener S' y R ,
 - $\delta \rightarrow R - \bar{R} + v(S', \omega) - v(S, \omega)$
 - $\bar{R} \rightarrow \bar{R} + \alpha \bar{R} \delta$
 - $\omega \rightarrow \omega + \alpha \omega \delta \nabla v(S, \omega)$
 - $\theta \rightarrow \theta + \alpha \theta \delta \nabla \ln \pi(A|S, \theta)$
 - $S \rightarrow S'$
-

1.4. Algoritmos de Aprendizaje Profundo por Refuerzo

Vamos a ver una serie de algoritmos de DRL, combinando ambos paradigmas, y abordaremos su funcionamiento.

1.4.1. DQN

Los algoritmos Deep Q-Networks (DQN) [MKS⁺13], son métodos basados en valor que sustituyen la tabla empleada en el algoritmo Q-learning por una red neuronal.

Realizando algunas modificaciones en el algoritmo Q-learning, construimos el siguiente **algoritmo inicial DQN**:

Algorithm 2: Algoritmo inicial DQN, Deep Q-Networks

1. Inicializar $Q(s, a)$ con valores aleatorios
 2. Interactuar con el entorno, obtener la tupla (s, a, r, s')
 3. Calcular el valor de la función de pérdida $\mathcal{L} = (Q(s, a) - r)^2$ si el episodio ha finalizado o $\mathcal{L} = (Q(s, a) - (r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')))^2$ si no ha finalizado.
 4. Volver al segundo paso hasta converger
-

Partiendo de este algoritmo inicial DQN, se construye un algoritmo final DQN que utiliza un método epsilon-greedy y gracias a ello nos ayuda a explorar el entorno al principio y a ampliar la muestra de experiencias. La solución principal se basa justo en ese algoritmo, existen otras soluciones a este problema y a día de hoy es una de las preguntas abiertas fundamentales en el aprendizaje por refuerzo además de un área de investigación activa que está lejos de resolverse por completo.

Estamos tratando de aproximar una función compleja y no lineal, $Q(S, A)$, con una red neuronal. Para la optimización con SGD debe cumplirse que las muestras de datos sean independientes e idénticamente distribuidas, pero esto no ocurre y tendremos que lidiar con ello. Para solucionarlo necesitamos utilizar un gran bufer de nuestra experiencia pasada y muestrear datos de entrenamiento de él, en lugar de utilizar nuestra experiencia más reciente. Esta técnica se llama bufer de repetición.

Para conseguir un entrenamiento de DQN mas estable y eficiente, se utilizó el método ϵ -greedy, el bufer de repetición y la red neuronal profunda que aproxima las tablas Q-Learning. La empresa de inteligencia artificial DeepMind entrenó con éxito un DQN en un conjunto de 49 juegos de Atari y demostró la eficiencia de este enfoque cuando se aplicó a entornos complicados. A continuación, se ilustra los pasos del **algoritmo DQN**:

Algorithm 3: DQN, Deep Q-Networks

1. Inicializar los parámetros de $Q(s, a)$ y $\bar{Q}(s, a)$ con pesos aleatorios, $\epsilon \rightarrow 1.0$, y vaciar el bufer de repetición
 2. Con probabilidad ϵ , seleccionar una acción aleatoria, a ; de lo contrario, $a = \arg \max_a Q(s, a)$
 3. Ejecutar la acción a sobre el entorno y observar la recompensa que se produce r' y el siguiente estado s' .
 4. Almacenar la transición (s, a, r, s') en el bufer de repetición.
 5. Muestrear un mini-lote aleatorio de transiciones del bufer de repetición.
 6. Para cada transición en el bufer, calcular el objetivo $y = r$ si el episodio ha terminado en este caso, o $y = r + \gamma \max_{a' \in \mathcal{A}} \bar{Q}(s', a')$ en otro caso.
 7. Calcular la pérdida: $\mathcal{L} = (Q(s, a) - y)^2$
 8. Actualizar $Q(s, a)$ usando el algoritmo SGD para minimizar la pérdida respecto al modelo paramétrico.
 9. Cada N step, copiar los pesos de Q a $\bar{Q}$.
 10. Volver al paso dos hasta converger
-

1.4.2. A2C

El algoritmo Advantage Actor Critic, (A2C) [MPBM⁺16], se encuentra dentro de los métodos actor-crítico donde tenemos un actor encargado de aprender una política mientras un crítico se encarga de evaluarla.

De esta forma, tenemos dos redes neuronales, una red “crítico” encargada de aprender el valor de los estados mapeando cada uno a sus valores correspondientes y una red “actor” que mapea cada estado a una distribución de probabilidad que nos devuelve la probabilidad de cada acción, por tanto la acción correspondiente con dicho estado será la que tenga una probabilidad mayor. Básicamente, el “actor” aprende la política $\pi(a|s)$ y el “crítico” la función valor $V(s)$.

A3C, definida como Asynchronous Advantage Actor Critic, implementa un entrenamiento paralelo donde varios trabajadores en entornos paralelos actualizan de manera independiente una función de valor global, de ahí el término “asíncrono”. A2C es una versión síncrona de A3C, que cuenta con un único agente coordinando las interacciones con el entorno. La coordinación la realiza esperando a que todos los nodos trabajadores terminen su trabajo antes de actualizar la función valor global.

A continuación, presentamos el pseudocódigo de este algoritmo:

Algorithm 4: A2C, Advantage Actor Critic

1. Inicializar los parámetros de la red, θ , con valores aleatorios.
 2. Ejecutar N pasos (steps) en nuestro entorno usando la actual política, π_θ , y obtener el estado, s_t , la acción, a_t , y la recompensa, r_t .
 3. $R = 0$ si se ha alcanzado el final del episodio
 4. Para $i = t - 1, \dots, t_{start}$
 - a) $R \rightarrow r_i + \gamma R$
 - b)
 - Acumular el gradiente de políticas
$$\partial\theta_\pi \rightarrow \partial\theta_\pi + \nabla_\theta \log \pi_\theta(a_i|s_i)(R - V_\theta(s_i))$$
 - Acumular el gradiente de funciones de valor:
$$\partial\theta_v \rightarrow \partial\theta_v + \frac{\partial(R - V_\theta(s_i))^2}{\partial\theta_v}$$
 - c) Actualizar los parámetros de la red usando los gradientes acumulativos, moviéndome en la dirección de la política del gradiente, $\partial\theta_\pi$, y en la dirección contraria del gradiente de la función de valor, $\partial\theta_v$
 - d) Volver al paso 2 hasta lograr converger
-

1.4.3. DDPG

Deep Deterministic Policy Gradient, (DDPG) [LHP⁺19], es un algoritmo *off-policy* del tipo actor-crítico que combina DPG (Deterministic Policy Gradient) con DQN.

Este algoritmo es de la familia de algoritmos de A2C pero la política es determinística, lo que significa que directamente nos proporciona la acción a tomar a partir del estado.

En la red del actor, tomará como entrada el estado y devolverá N valores, uno por cada acción. El mapeo será determinista, ya que la misma red siempre devolverá la misma salida si la entrada es la misma.

Por otro lado, el crítico estimará el valor Q sabiendo que nuestra acción es un vector de números, entonces tenemos dos entradas: el estado y la acción. Su salida será un único valor que se corresponde con el valor de la función Q . Esta arquitectura difiere de la de DQN ya que ésta devolvía valores para todas las acciones a la vez cuando nuestro espacio de acciones era discreto.

Luego, tenemos dos funciones:

1. El actor que lo denotamos como $\mu(s)$
2. El crítico denotado como $Q(s, a)$, mas bien, $Q(s, \mu(s))$.

En el teorema del gradiente de política determinística se demostró que el gradiente de política estocástica es equivalente al gradiente de política determinística. Así que, para mejorar la política, solo necesitamos calcular el gradiente de $Q(s, \mu(s))$ y aplicando la regla de la cadena se obtiene:

$$\nabla Q(s, \mu(s)) = \nabla_a Q(s, a) \nabla_{\theta_\mu} \mu(s).$$

Algorithm 5: DDPG, Deep Deterministic Policy Gradient

1. Inicializar aleatoriamente la red del crítico $Q(s, a|\theta_Q)$ y del actor $\mu(s|\theta_\mu)$ con pesos θ_Q y θ_μ
2. Inicializar la red etiqueta Q' y μ' con pesos $\theta_{Q'} \rightarrow \theta_Q$, $\theta_{\mu'} \rightarrow \theta_\mu$
3. Inicializar el bufer R
4. Para $EPISODIO = 1, \dots, M$ hacer
 - a) Inicializamos un proceso aleatorio $\mathcal{N}$ para explorar acciones
 - b) Observación inicial s_1
 - c) Para $t = 1, \dots, T$ HACER
 - 1) Seleccionar la acción $a_t = \mu(s_t|\theta_\mu) + \mathcal{N}_t$ de acuerdo a la política actual y a un ruido de exploración.
 - 2) Ejecutar la acción a_t y observar la recompensa r_t y el nuevo estado s_{t+1}
 - 3) Almacenamos la transición $(s_t, a_t, r_t, s_{t+1}) \in R$
 - 4) Muestreamos un minibatch aleatorio de N transiciones $(s_i, a_i, r_i, s_{i+1}) \in R$
 - 5) Ajustamos $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta_{\mu'}))|_{\theta_{Q'}}$
 - 6) Actualizamos el crítico minimizando la pérdida $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta_Q))^2$
 - 7) Actualizamos la política del actor usando gradiente de la política de la muestra:

$$\nabla_{\theta_\mu} J \sim \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta_Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta_\mu \mu(s|\theta_\mu)}|_{s_i}$$

- 8) Actualizar las etiquetas de las redes

$$\theta_{Q'} \rightarrow \tau \theta_Q + (1 - \tau) \theta_{Q'}$$

$$\theta_{\mu'} \rightarrow \tau \theta_\mu + (1 - \tau) \theta_{\mu'}$$

1.4.4. TRPO

El algoritmo Trust Region Policy Optimization (TRPO) [KL02], propuesto en 2015, se utiliza para mejorar políticas de un agente de aprendizaje por refuerzo de manera eficiente y segura. A diferencia de otros métodos que realizan actualizaciones de políticas, este algoritmo introduce una restricción adicional en la actualización de la política para así poder controlar cuánto puede cambiar la política en cada iteración. Esta restricción se expresa en términos de la divergencia de Kullback-Leibler (KL) entre la política anterior y la nueva política propuesta.

Sea π la política anterior con pesos θ_{Old} y la nueva política $\bar{\pi}$ con pesos θ , definimos la restricción adicional en la actualización de la política, esto es, la divergencia KL máxima entre las políticas antigua y nueva, como: $\bar{D}_{KL}^{\theta_{Old}}(\theta_{Old}, \theta) \leq 0$.

El método de TRPO define la siguiente función que indica la **frecuencia de visita del estado** s : $\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \dots$. En esta ecuación, $P(s_i = s)$ es igual a la probabilidad de muestrear la posición i de la trayectoria con el estado s .

Entonces, TRPO define la siguiente **función objetivo** a optimizar

$$\mathcal{L}_\pi(\bar{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \bar{\pi}(a|s) A_\pi(s, a)$$

donde $\mu(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t)]$ es la **recompensa de descuento esperada de la política** π y $\bar{\pi} = \arg \max_a A_\pi(s, a)$ define la **política determinística** y la **función ventaja** se define como

$$A_\pi = Q_\pi(s, a) - V_\pi(s) \quad (1.6)$$

dónde $Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} [\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})]$ y $V_\pi(s_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} [\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})]$, $a_t \sim \pi(a_t|s_t)$, $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$, $s_0 \sim p_0(s_0)$ donde $p_0 : \mathcal{S} \rightarrow \mathbb{R}$ es la distribución de probabilidad del estado inicial s_0 , $r : \mathcal{S} \rightarrow \mathbb{R}$ es la función recompensa, y $\gamma \in (0, 1)$ es el factor de descuento.

Como tendremos una política parametrizada, π_θ , donde $\pi_\theta(a|s)$ es una función diferenciable respecto al vector de parámetros θ , entonces podemos calcular el gradiente de la función objetivo. En [KL02] se descubrió que $\mathcal{L}_\pi$ coincidía con $\eta(\pi)$ para el primer valor θ_0 . Consecuentemente, tenemos

$$\begin{aligned} \mathcal{L}_{\pi_{\theta_0}}(\pi_{\theta_0}) &= \eta(\pi_{\theta_0}) \\ \nabla_\theta \mathcal{L}_{\pi_{\theta_0}}(\pi_\theta)|_{\theta=\theta_0} &= \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_0} \end{aligned} \quad (1.7)$$

La ecuación 1.7 nos dice que para tamaños de pasos muy pequeños $\pi_{\theta_0} \rightarrow \bar{\pi}$ y el paper [KL02] propone la siguiente actualización de políticas, definida como actualización iterativa de políticas conservativas,

$$\pi_{new}(a|s) = (1 - \alpha)\pi_{old}(a|s) + \alpha \pi'(a|s)$$

$$\eta(\pi_{new}) \geq \mathcal{L}_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2 \quad (1.8)$$

donde $\pi'(a|s) = \arg \max_{\pi'} \mathcal{L}_{\pi_{old}}(\pi')$ y $\epsilon = \max_s |\mathbb{E}_{A \sim \pi'(a|s)}[A_\pi(s, a)]|$. Para extender esta idea a políticas estocásticas generales tenemos que realizar un cambio en la cota que se le ha impuesto a $\eta(\pi_{new})$ y en la constante ϵ . Vamos a utilizar la divergencia variacional total, definida como $D_{TV}(p||q) = \frac{1}{2} \sum_i |p_i - q_i|$ para cualquier distribución de probabilidad p y q . Por tanto, podemos definir, $D_{TV}(\pi||\bar{\pi}) = \max_s D_{TV}(\pi(\cdot|s)||\bar{\pi}(\cdot|s))$ y si $\alpha = D_{TV}^{max}(\pi_{old}||\pi_{new})$, entonces tenemos la siguiente cota para $\eta(\pi_{new})$ donde $\epsilon = \max_{s,a} |A_\pi(s, a)|$.

$$\eta(\pi_{new}) \geq \mathcal{L}_{\pi_{old}}(\pi_{new}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2 \quad (1.9)$$

La divergencia variacional total y la divergencia KL presentan la siguiente relación: $D_{TV}(p||q)^2 \leq D_{KL}(p||q)$ para cualquier distribución de probabilidad p y q . Aplicando esto a la ecuación 1.9, podemos obtener una nueva cota sabiendo que $D_{KL}^{max}(\pi, \bar{\pi}) = \max_s D_{KL}(\pi(\cdot|s), \bar{\pi}(\cdot|s))$ y $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$

$$\eta(\pi_{new}) \geq \mathcal{L}_\pi(\bar{\pi}) - CD_{KL}^{max}(\pi, \bar{\pi}) \quad (1.10)$$

Por consiguiente, se obtiene el siguiente algoritmo que va iterando por políticas garantizando que el valor η no decrezca.

Algorithm 6: TRPO, Iteración de políticas

1. Inicializar π_0
2. Para $i = 0, 1, 2, \dots$ hasta converger hacer
 - a) Calcular todos los valores de la función ventaja $A_{\pi_i}(s, a)$
 - b) Resolver un problema de optimización con restricciones:

$$\pi_{i+1} = \arg \max_{\pi} [\mathcal{L}_{\pi_i}(\pi) - CD_{KL}^{max}(\pi_i, \pi)]$$

donde $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$ y $\mathcal{L}_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \bar{\pi}(a|s) A_{\pi_i}(s, a)$

Hemos considerado el problema de optimización de políticas independiente de la parametrización de π . Ahora vamos a ver como traducir el algoritmo anterior para utilizar políticas parametrizadas π_θ . Usaremos la siguiente notación, $\eta(\theta) := \eta(\pi_\theta)$ y $\mathcal{L}_\theta(\bar{\theta}) := \mathcal{L}_{\pi_\theta}(\pi_{\bar{\theta}})$ y $D_{KL}(\theta||\bar{\theta}) := D_{KL}(\pi_\theta||\pi_{\bar{\theta}})$

Atendiendo a esta notación, nuestro problema de optimización con restricción del algoritmo se traduce a $\max_{\theta} [\mathcal{L}_{\theta_{Old}}(\theta) - CD_{KL}^{max}(\theta_{Old}, \theta)]$ pero podemos darnos cuenta que maximizar esta expresión es equivalente a maximizar la función objetivo sujeto a una restricción de la región verdadera:

$$\max_{\theta} \mathcal{L}_{\theta_{Old}}(\theta) \text{ sujeto a } D_{KL}^{max}(\theta_{Old}, \theta) \leq \delta$$

Este problema se encuentra motivado teóricamente pero en la práctica es intratable debido al gran número de restricciones. Sin embargo, podemos utilizar la media de las divergencias KL, $\bar{D}_{KL}^{\rho_{\theta_{Old}}}(\theta_{old}, \theta) := \mathbb{E}_{s \sim p}[D_{KL}(\pi_{\theta_1(\cdot|s)} || \pi_{\theta_2(\cdot|s)})]$, y resolver el siguiente problema de optimización y así ir actualizando las políticas.

$$\max_{\theta} \mathcal{L}_{\theta_{Old}}(\theta) \text{ sujeto a } \bar{D}_{KL}^{\rho_{\theta_{Old}}}(\theta_{Old}, \theta) \leq \delta$$

1.4.5. PPO

Los algoritmos Proximal Policy Optimization (PPO) [SWD⁺17], fue propuesto en 2017, no mucho después de TRPO, y sin embargo es mucho mas simple que TRPO. Igualmente, corresponde a la familia de métodos que calculan gradientes de políticas. PPO supera a otros métodos de gradientes de políticas online, y establece un balance general o punto intermedio entre complejidad y simplicidad del algoritmo.

En la literatura se ha visto que PPO ha alcanzado resultados muy notables cuando se comparó con otros algoritmos, de ahí surgió mi motivación de usar PPO en mi experimentación en lugar de TRPO.

Los métodos de gradientes de políticas se encargan de calcular una estimación del gradiente de la política, $g = \mathbb{E}_t[\nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \bar{A}_t]$ donde π_{θ} es una política estocástica y $\bar{A}_t$ es un estimador de la función ventaja (Ecuación 1.6) en el timestep t , y utilizan g en el algoritmo del gradiente descendente estocástico.

La **estimación de la función ventaja** en el timestep t , $\bar{A}_t$, se define como

$$\bar{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

donde $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

donde $t \in [0, T]$, T es la longitud de la trayectoria y λ se suele ajustar a 1 en muchos casos prácticos.

Tenemos la siguiente función objetivo cuyo gradiente es la estimación del gradiente de la política.

$$L^{PG}(\theta) = \mathbb{E}_t[\log \pi_{\theta}(a_t|s_t) \bar{A}_t]$$

En **TRPO** tratamos de **maximizar la función objetivo sujeto a una restricción** en el tamaño de la actualización de la política.

$$\begin{aligned} & \max_{\theta} \mathbb{E}_t\left[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{Old}}(a_t|s_t)} \bar{A}_t\right] \\ & \text{sujeto a } \mathbb{E}_t[KL[\pi_{\theta_{Old}}(\cdot|s_t), \pi_{\theta}(\cdot|s_t)]] \leq \delta \end{aligned}$$

Resulta que la teoría justifica que TRPO proponga resolver un problema de aproximación sin restricciones equivalente para algún coeficiente β :

$$\max_{\theta} \mathbb{E}_t\left[\frac{\pi_{\theta}}{\pi_{\theta_{Old}}} \bar{A}_t - \beta KL[\pi_{\theta_{Old}}(\cdot|s_t), \pi_{\theta}(\cdot|s_t)]\right]$$

Para lograr nuestro objetivo de obtener un algoritmo de primer orden que simule una mejora monótona de TRPO, necesitamos añadir modificaciones adicionales y ahí es donde surge **la función objetivo de PPO**, *clipped surrogated*, $L^{CPI}(\theta)$ que se intenta maximizar con respecto a θ .

$$L^{CPI}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\bar{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\bar{A}_t)] \quad (1.11)$$

donde ϵ es el hiperparámetro (suele ser 0.1 o 0.2), θ es el parámetro de la política, r_t es el ratio de probabilidad entre la nueva y la antigua política, $\bar{A}_t$ es la estimación de la función ventaja para el timestep t y clip es una función que modifica los valores r_t , eliminando los valores fuera del intervalo $[1 - \epsilon, 1 + \epsilon]$.

Mostramos la **motivación de como surge la función objetivo de PPO** y después el **pseudocódigo del algoritmo**. La idea de PPO surgió de combinar conceptos de A2C y TRPO. Veamos cómo paso a paso llegamos a obtener la función objetivo de PPO, esto es, L^{CPI}

- Respecto a A2C, introduce una primera mejora que consiste en modificar la formula que utilizamos para estimar los gradientes de la política. En lugar de utilizar el gradiente del logaritmo de la probabilidad de que una acción sea escogida, el método PPO utiliza como función objetivo el ratio de proporción de mejora entre la nueva política y la antigua.

Así, tenemos que el antiguo objetivo de A2C es $J_\theta = \mathbb{E}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\bar{A}_t]$ y que PPO propone cambiarlo a $J_\theta = \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{Old}}(a_t|s_t)}\bar{A}_t]$.

- Si comenzamos a maximizar este valor, nos damos cuenta que se produce una actualización muy grande de los pesos de la políticas, es decir, grandes desviaciones en la actualización de políticas. Para limitar esto, se propone una función objetivo donde el ratio queda en el intervalo $1 - \epsilon$ y $1 + \epsilon$.

El ratio entre la nueva y la vieja política se define como $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{Old}}(a_t|s_t)}$. Este ratio se emplea a la función objetivo que hemos mencionado anteriormente J_θ y con la definición de $r_t(\theta)$, podemos reescribir $J_\theta = \mathbb{E}_t[r_t(\theta)\bar{A}_t]$

- Aún no hemos hablado de la función $\bar{A}_t$, PPO tratará de estimar la función ventaja y esta será la segunda mejora respecto a A2C. $\bar{A}_t$ es conocida como la estimación de la función acción-ventaja y los paper de A2C se encargan de estimarlo en un horizonte finito de estimaciones de T pasos de la forma $\bar{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{T-t+1}r_{T-1} + \gamma^{T-t}V(s_t)$. El paper propuesto de PPO propone definir $\bar{A}_t = \sigma_t + (\gamma\alpha)\sigma_{t+1} + (\gamma\alpha)^2\sigma_{t+2} + \dots + (\gamma\lambda)^{T-t+1}\sigma_{t-1}$ donde $\sigma_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

- Finalmente, nuestra función de pérdida sería:

$$L^{CLIP}(\theta) = \mathbb{E}[\min(r(\theta)\bar{A}_{\theta_{old}}(s, a), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)\bar{A}_{\theta_{old}}(s, a))]$$

Algorithm 7: PPO, Proximal Policy Optimization

1. Para $i=1,2,\dots$ hacer
 - a) Para $j=1,2,\dots,N$ hacer
 - 1) Ejecutar la política $\pi_{\theta_{Old}}$ en el entorno con T pasos (steps)
 - 2) Calcular las estimaciones de la función ventajas $\bar{A}_1, \bar{A}_2, \dots, \bar{A}_T$
 - b) fin
 - c) Optimizar el objetivo clipped surrogated L^{CPI} , con K épocas y tamaño de minibatch $M \leq NT$
 - d) $\theta_{Old} \rightarrow \theta$
-

1.4.6. SAC

El algoritmo Soft Actor Critic (SAC) [HZAL18], trata de implementar un método *off-policy* del tipo actor-critic cuyo fin es optimizar políticas estocásticas en entornos continuos. Fue propuesto por un grupo de investigadores de Berkeley que introdujeron esta idea en un artículo publicado en 2018. Hasta el momento, se considera uno de los mejores métodos de control en problemas continuos.

Su idea central es la regularización de la entropía, con el objeto de entrenar la política para obtener un equilibrio entre la recompensa esperada y el valor de entropía. El valor de la entropía nos marca una medida de aleatoriedad del comportamiento del agente mientras que el valor de la recompensa nos indica el comportamiento exacto que debe seguir el Agente para los estados del entorno. Establecer esta regularización es equivalente a cuando en otros algoritmos hablábamos de la necesidad de alcanzar un equilibrio entre la exploración y la explotación.

La política que busca el algoritmo es la siguiente, donde H es la medida de entropía y R la recompensa:

$$\pi^* = \arg \max_a \mathbb{E}_{\tau \sim \pi} [\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t)))]$$

SAC hace uso de cuatro redes neuronales para llevar a cabo el objetivo de optimización que se acaba de presentar.

- Dos redes Q-networks
- Una red V-networks
- Una red de políticas, π_θ , entrenada con DDPG.

Algorithm 8: SAC, Soft Actor Critic

1. Inicializar los parámetros $\psi, \bar{\psi}, \theta, \phi$
 2. Para cada iteración hacer
 - a) Para cada paso del entorno hacer
 - 1) $a_t \sim \pi_\phi(a_t|s_t)$
 - 2) $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
 - 3) $\mathcal{D} \rightarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
 - b) fin
 - c) Para cada paso del gradiente hacer
 - 1) $\psi \rightarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$
 - 2) $\theta_i \rightarrow \theta_i - \lambda_Q \nabla_{\theta_i} J_Q(\theta_i)$ para $i \in \{1, 2\}$
 - 3) $\phi \rightarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$
 - 4) $\bar{\psi} \rightarrow \tau\psi + (1 - \tau)\bar{\psi}$
 - d) fin
-

2. Redes Generativas Adversarias (GANs)

En este capítulo se expondrá una aproximación de un modelo generativo para la generación de contenido sintético con redes neuronales. Aunque los modelos generativos presentan diversos enfoques y dominios de aplicación, en este trabajo nos centraremos en su uso en Aprendizaje por Refuerzo. Se han analizado en profundidad las documentaciones [LB19], [PC08] [Nie19], [Wan20], [Goo17] y [GPAM⁺14].

2.1. Fundamentos

2.1.1. Concepto

Las redes generativas adversarias (*Generative Adversarial Networks*, *GANs*) son una aproximación de tipo generativo basado en redes neuronales diferenciables. Se basan en el planteamiento seguido en teoría de juegos donde la red generadora debe competir contra su adversario, la red discriminadora. La red generadora produce muestras $x = G(z; \theta^{(G)})$ y su adversario, la red discriminadora, intenta distinguir entre las muestras extraídas del conjunto de entrenamiento y muestras construidas por el generador. El discriminador se encarga de asignarle una probabilidad a la entrada x , y $D(x; \theta^{(D)})$ indica la probabilidad de que x sea un ejemplo de entrenamiento real en lugar de una muestra sintética (los parámetros $\theta^{(G)}$, $\theta^{(D)}$ serán aprendidos en el entrenamiento de la GAN).

Por tanto, las GANs son una clase de técnicas de aprendizaje automático que consiste en dos modelos que se entrenan de forma simultánea: la red del generador entrenada para generar datos sintéticos y la red del discriminador entrenada para distinguir las muestras falsas de muestras reales.

- El término *generativo* se refiere al objetivo central del modelo, que es generar nuevos datos.
- El término *adversario* apunta a la competitividad que existe en este planteamiento de teoría de juegos, donde compiten los dos modelos: el generador y el discriminador. El objetivo del generador es crear nuevos datos que no se puedan distinguir de los datos reales del conjunto de entrenamiento. Las dos redes intentan siempre competir entre ellas: cuanto mejor se vuelve el generador para crear nuevos datos similares a los datos del conjunto de entrenamiento, mejor debe ser el discriminador para distinguir los datos reales de los sintéticos.
- El término *redes* indica la clase de modelos de aprendizaje automático que se usa para representar al generador y al discriminador: las redes neuronales.

2.1.2. Funcionamiento

Nos basamos en el ejemplo de generación de dígitos escritos a mano para explicar el sistema de acción de las GANs. Vamos a estudiar los detalles enumerados del siguiente diagrama extraído de [SWD⁺17].

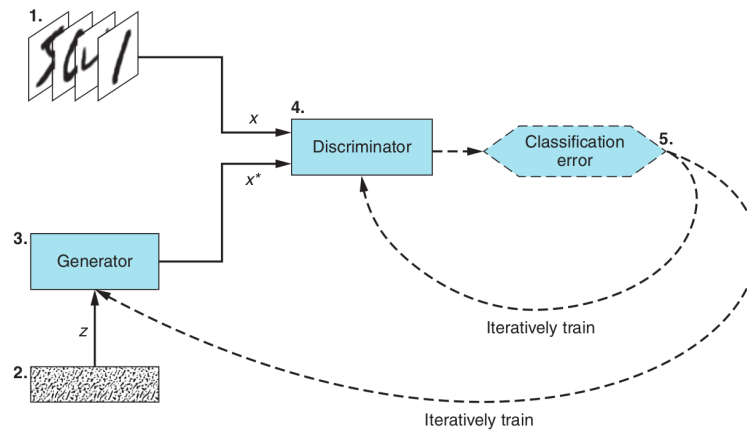


Figura 2.1.: Funcionamiento de una GANs [LB19]

1. Conjunto de datos de entrenamiento

Es el conjunto de datos de ejemplos reales que queremos que el generador aprenda para que genere muestras sintéticas, x^* , de alta calidad. Sirve como entrada a la red discriminadora y, en este caso, el conjunto de datos consta de imágenes de dígitos escritos a mano.

2. Vector de ruido aleatorio

Es la entrada a la red del generador, z , y consiste en un vector de números aleatorios que el generador usa como punto de partida para sintetizar ejemplos sintéticos.

3. Red del generador

El generador recibe como entrada un vector de números aleatorios, z , y genera ejemplos sintéticos, x^* . Su objetivo es que no se puedan distinguir los ejemplos sintéticos que produce de los ejemplos reales del conjunto de datos de entrenamiento.

4. Red del discriminador

El discriminador recibe como entrada un ejemplo real, x , proveniente del conjunto de entrenamiento o un ejemplo falso, x^* , producido por el generador. En cada ejemplo, el discriminador determina y genera la probabilidad de que el ejemplo sea real.

5. Entrenamiento/ajuste iterativo

Para cada una de las predicciones del discriminador, determinamos los aciertos y

errores de las probabilidades que asigna a cada muestra, como lo haríamos con un clasificador normal, y usamos los resultados para ajustar iterativamente las redes del discriminador y del generador a través de la retropropagación:

- Los pesos y sesgos del discriminador se actualizan para maximizar su exactitud (accuracy) de clasificación. De esta forma maximiza la probabilidad de predicción correcta: x como real y x^* como falso.
- Los pesos y sesgos del generador se actualizan para maximizar la probabilidad de que el discriminador clasifica erróneamente, esto es, intenta minimizar su exactitud (accuracy) para predecir x^* como real.

Básicamente, se entrenan las dos redes mediante retropropagación utilizando la pérdida del discriminador. El discriminador se esfuerza por minimizar la pérdida tanto para los ejemplos reales como para los sintéticos, mientras que el generador intenta maximizar la pérdida del discriminador para los ejemplos sintéticos que produce.

Resumiendo el funcionamiento en dos ideas básicas, tenemos que primero vamos a desarrollar un modelo generador que convierte un vector en una imagen ya que en este ejemplo nuestra base de datos esta formada por imágenes y a continuación, un modelo discriminador toma como entrada una imagen candidata (real de la base de datos o sintética generada) y la clasifica en una de las dos clases.

2.2. Entrenamiento de GANs

El algoritmo de entrenamiento GAN tiene dos partes principales. Estas dos partes, el entrenamiento del discriminador y el entrenamiento del generador, representan la misma red GAN en diferentes instantes de tiempo en las etapas correspondientes al proceso de entrenamiento. El siguiente diagrama Figura 2.2 detalla el entrenamiento de una GANs cuya base de datos son dígitos escritos a mano, pero podría abstraerse a cualquier base de datos de imágenes.

A continuación, veamos de forma esquematizada el proceso de entrenamiento.

Algorithm 9: Entrenamiento del generador

1. Obtenemos un vector de ruido z , y usamos la red del generador para sintetizar un ejemplo falso x^* .
 2. Usamos la red del discriminador para clasificar x^* .
 3. Calculamos el error de clasificación y retro-propagamos el error para actualizar los parámetros entrenables del generador, buscando **maximizar** el error del discriminador.
-

Algorithm 10: Entrenamiento del discriminador

1. Tomamos un ejemplo real aleatorio x del conjunto de datos de entrenamiento.
2. Obtenemos un vector de ruido z , y usando la red del generador generamos un ejemplo falso x^* .
3. Usamos la red del discriminador para clasificar x y x^* .
4. Calculamos el error de clasificación y retro-propagamos el error para actualizar los parámetros entrenables del discriminador, buscando **minimizar** el error del discriminador.

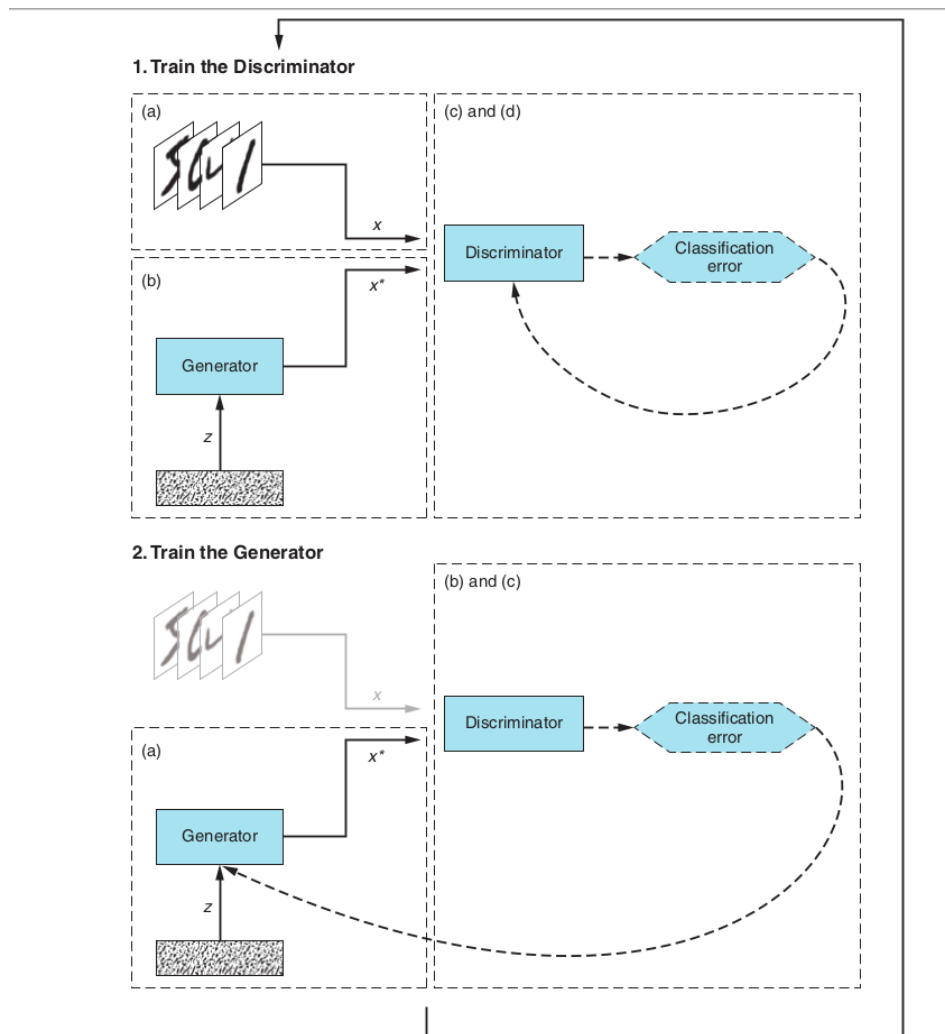


Figura 2.2.: Entrenamiento de una GANs [LB19]

Es muy importante tener en cuenta que hay que configurar el discriminador para que se congelen sus pesos durante el entrenamiento del generador. Ya que si los pesos del discriminador pudieran actualizarse durante este proceso, entonces estaría entrenando al discriminador para que siempre clasifique como imagen real.

En este momento, nos cuestionamos cuando debe detenerse el ciclo de entrenamiento GAN, ¿cómo sabemos cuando una GAN está completamente entrenada para que podamos determinar el número de iteraciones de entrenamiento? Con una red neuronal normal tenemos un objetivo específico, por ejemplo con un clasificador medimos el error de clasificación en los conjuntos de entrenamiento y validación, y detenemos el proceso cuando el error de validación comienza a empeorar para evitar que se produzca sobreajuste. En una GAN, las dos redes tienen objetivos contrapuestos: cuando una red mejora, la otra empeora. Entonces nos preguntamos cómo determinamos cuando parar el entrenamiento.

En teoría de juegos puede conocerse esta configuración como un juego de suma cero y todos estos juegos tienen un equilibrio de Nash, un punto en el que ningún jugador puede mejorar su situación o recompensa cambiando sus acciones.

Imaginemos que los dos jugadores en el juego están representados por dos funciones, cada una de las cuales es diferenciable tanto con respecto a sus entradas como con respecto a sus parámetros. El discriminador es una función D que toma x como entrada y usa $\theta^{(D)}$ como parámetros. Asimismo, el generador está definido por una función G que toma z como entrada y utiliza $\theta^{(G)}$ como parámetros. Ambos jugadores tienen funciones de costos que se definen en término de los parámetros de ambos jugadores. El discriminador desea minimizar $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ y debe hacerlo controlando solo $\theta^{(D)}$. De igual forma, el generador desea minimizar $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ y debe hacerlo controlando solo $\theta^{(G)}$. Debido a que el costo de cada jugador depende de los parámetros del otro jugador, pero cada jugador no puede controlar los parámetros del otro jugador, este escenario es más sencillo de describir como un juego que como un problema de optimización. La solución a un problema de optimización es un mínimo (local) pero la solución de un juego es un equilibrio Nash, esto es, una tupla $(\theta^{(D)}, \theta^{(G)})$ que es un mínimo local de $J^{(D)}$ con respecto a $\theta^{(D)}$ y un mínimo local de $J^{(G)}$ con respecto a $\theta^{(G)}$.

Una GAN alcanza el equilibrio de Nash cuando se cumplen las siguientes condiciones:

- El generador produce ejemplos sintéticos que son indistinguibles de los ejemplos reales en el conjunto de datos de entrenamiento.
- El discriminador puede adivinar aleatoriamente si un ejemplo en particular es real o falso.

Pensemos que cuando se alcanza el equilibrio, el discriminador no tiene ninguna herramienta para poder distinguir un ejemplo falso de uno verdadero. Por tanto, debido a que la mitad de los ejemplos que recibe son reales y la otra mitad sintéticos, lo mejor que puede hacer el discriminador es lanzar una moneda al aire y clasificar cada ejemplo como real o falso con 50 % de probabilidad. De esta forma, se dice que la GAN ha convergido, pero en la práctica es casi imposible encontrar el equilibrio Nash debido a las complejidades que tiene la convergencia en juegos no convexos.

2.2.1. Funciones de coste

Siguiendo la notación estándar presentada anteriormente, sea $J^{(G)}$ la función de costo del generador y $J^{(D)}$ la función de costo del discriminador. Los parámetros entrenables de las dos redes los denotamos como: $\theta^{(G)}$ para el generador, $\theta^{(D)}$ para el discriminador. Por tanto, tenemos

- Función de costo del generador: $J^{(G)}(\theta^{(D)}, \theta^{(G)})$
- Función de costo del discriminador: $J^{(D)}(\theta^{(D)}, \theta^{(G)})$

El equilibrio de Nash se puede matematizar siguiendo esta notación; esto es, el equilibrio de Nash ocurre cuando el generador minimiza la función $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ con respecto a los parámetros entrenables del generador $\theta^{(G)}$ y, simultáneamente, la función de costo del discriminador $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ se minimiza con respecto a los parámetros entrenables del discriminador $\theta^{(D)}$.

Con esta terminología el algoritmo de entrenamiento queda de la siguiente forma:

Algorithm 11: Entrenamiento de una GAN

Por cada iteración del entrenamiento **hacer**

1. Entrenar al discriminador
 - a) Tomar un mini lote de ejemplos reales: x
 - b) Tome un mini lote de vectores de ruido aleatorio z y genere un mini lote de ejemplos sintéticos: $G(z) = x^*$
 - c) Calcule la pérdida de clasificación para $D(x)$ y $D(x^*)$, y retropropague el error total para actualizar $\theta^{(D)}$ para **minimizar** la pérdida de clasificación.
2. Entrenar al generador
 - a) Tome un mini lote de vectores aleatorios de ruidos z y genere un mini lote de ejemplos sintéticos: $G(z) = x^*$.
 - b) Calcule la pérdida de clasificación para $D(x^*)$, y retropropague la pérdida para actualizar $\theta^{(G)}$ para **maximizar** la pérdida de clasificación.

Fin

El proceso de entrenamiento consta de dos SGD simultáneos. En cada paso, se muestran dos mini lote: un mini lote de valores x del conjunto de datos y un mini lote de valores z . Luego se realizan dos pasos del algoritmo de gradiente descendente de forma simultánea: uno que actualiza $\theta^{(D)}$ para reducir $J^{(D)}$ y otro que actualiza $\theta^{(G)}$ para reducir $J^{(G)}$. En ambos casos, usar Adam suele ser una buena opción. Por otra parte, aunque muchos autores recomendaron ejecutar más pasos de un jugador que del otro jugador, en 2016 se afirmó [Goo17] que el protocolo que mejor funciona en la práctica es el descenso de gradiente simultáneo, con un paso para cada jugador. A continuación,

procedemos a especificar las funciones de costo de las dos redes. Se pueden usar varias funciones de costos diferentes dentro del marco de las GAN.

1. Función de coste del discriminador

En la literatura, todos los diferentes juegos diseñados para GAN usan el mismo costo para el discriminador, $J^{(D)}$. Difieren solo en términos del costo utilizado para el generador, $J^{(G)}$. El costo utilizado para el discriminador es:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x \sim P_X} \log D(x) - \frac{1}{2}\mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$$

Esta función es justo el costo de la *entropía cruzada* que se minimiza cuando se entrena un clasificador binario estándar con una salida sigmoidea.

2. Minmax

La versión más simple del juego es un juego de suma cero, en el que la suma de todos los costos del jugador son siempre cero. Por tanto, la función de **costo del generador** sería

$$J^{(G)} = -J^{(D)}$$

Debido a que $J^{(G)}$ está ligado directamente a $J^{(D)}$, podemos resumir todo el juego con un valor de función que especifica el pago del discriminador:

$$v(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$$

Los juegos de suma cero también se denominan juegos minmax porque su solución implica la minimización en un bucle externo y la maximización en un bucle interno:

$$(\theta^{(D)}, \theta^{(G)})^* = \arg \min_{\theta^{(D)}} \max_{\theta^{(G)}} v(\theta^{(D)}, \theta^{(G)})$$

3. Motivación heurística

El costo utilizado para el generador en el juego minmax es útil para el análisis teórico, pero no funciona especialmente bien en la práctica.

En el juego minmax, el discriminador minimiza la entropía cruzada, pero el generador maximiza la misma entropía cruzada. Esto no es óptimo para el generador, porque cuando el discriminador rechaza con éxito las muestras del generador con alta confianza, el gradiente del generador se desvanece.

Para resolver este problema, un enfoque es continuar utilizando la minimización de entropía cruzada para el generador. En lugar de cambiar el signo del costo del discriminador para obtener un costo para el generador, cambiamos el objetivo utilizado para construir el costo de la entropía cruzada. El **costo para el generador** entonces se convierte en:

$$J^{(G)} = -\mathbb{E}_{z \sim P_Z} \log D(G(z))$$

En el juego minmax, el generador minimiza la probabilidad logarítmica de que el discriminador sea correcto. En este juego, el generador maximiza la probabilidad logarítmica de que el discriminador se equivoque.

2.2.2. Dificultades del aprendizaje

El aprendizaje de GAN puede ser difícil en la práctica cuando G , D son redes neuronales y $\max_D v(\theta^{(G)}, \theta^{(D)})$ no es convexo. De hecho, la falta de convergencia puede hacer que las GANs no se ajusten bien. En general, no se garantiza que el descenso simultáneo del gradiente en los costos de dos jugadores alcance un equilibrio. Consideresé, por ejemplo, la función $v(a, b) = ab$, donde un jugador controla una función de costo ab , mientras que el otro jugador controla el costo $-ab$. Si imaginamos a cada jugador haciendo pasos de gradiente infinitesimalmente pequeños, cada jugador reduciendo su propio costo a expensas del otro jugador, entonces a y b entran en una órbita circular estable, en lugar de llegar al punto de equilibrio en el origen. Tenga en cuenta que los equilibrios para un juego minmax no son mínimos locales de v . En cambio, son puntos que son simultáneamente mínimos para los costos de ambos jugadores. Esto significa que son punto silla de v que son mínimos locales con respecto a los parámetros del primer jugador y máximos locales con respecto a los parámetros del segundo jugador.

En experimentos realistas, la formulación de mejor rendimiento es una formulación que no es de suma cero. En esta formulación (véase la función de coste 3 en la sección 2.2.1) el generador tiene como objetivo aumentar la probabilidad logarítmica de que el discriminador cometa un error, en lugar de intentar disminuir la probabilidad logarítmica de que el discriminador haga la predicción correcta.

La estabilización del aprendizaje GAN sigue siendo un problema abierto. Afortunadamente, el aprendizaje GAN funciona bien cuando la arquitectura del modelo y los hiperparámetros se seleccionan cuidadosamente.

2.3. Fundamentos matemáticos de las GANs

Las GANs aparecieron como un nuevo modelo generativo cuyo objetivo era resolver la siguiente cuestión: tenemos un conjunto de datos de objetos con cierto grado de consistencia, por ejemplo, una colección de imágenes de perros o dígitos escritos a mano, o pinturas de Leonardo da Vinci, etc., ¿podemos generar artificialmente objetos similares?

En esta sección nos encargamos de formular esta pregunta y de responderla en base a conceptos matemáticos. Primero vamos a explicar que se entiende por “objetos con cierto grado de consistencia” u “objetos similares” antes de continuar el estudio. Debemos de suponer que nuestros objetos son puntos de $\mathbb{R}^n$, el conjunto de objetos lo denotamos como $X \subset \mathbb{R}^n$

Definimos las distribuciones de probabilidad $P_X : \mathcal{B}^n \rightarrow [0, 1]$ y $P_Y : \mathcal{B}^n \rightarrow [0, 1]$

cuyas funciones de densidad son $f_X : \mathbb{R}^n \rightarrow [0, 1]$ y $f_Y : \mathbb{R}^n \rightarrow [0, 1]$ para las variables aleatorias multidimensionales X e Y .

- Si decimos que los objetos en el conjunto de datos X tienen **cierto grado de consistencia**, queremos decir que son muestras generadas a partir de una misma distribución de probabilidad P_X en $\mathbb{R}^n$, donde f_X es su función de densidad. Al suponer que P_X tiene una función de densidad, entonces estamos afirmando que P_X es absolutamente continua.
- Explicar qué se entiende por **objetos similares** es un poco más complicado ya que hay muchas formas de cuantificar la similitud. Por ejemplo, podemos definir una función distancia y así decir que dos objetos son similares si la distancia entre ellos es pequeña. Pero esta idea no es útil aquí ya que nuestro objetivo no es generar nuevos objetos que tengan distancias pequeñas a algunos objetos del conjunto X . Más bien, queremos generar nuevos objetos que aunque no estén tan cerca de los objetos del conjunto de entrenamiento pero si pertenecerán a la misma clase, por ejemplo que sean pinturas de Leonardo da Vinci.

Un mejor enfoque es definir la similitud entre distribuciones de probabilidad. Es decir, tenemos nuestro conjunto de datos de entrenamiento $X \subset \mathbb{R}^n$ que esta formado a partir de muestras de una distribución de probabilidad P_X , con función de densidad f_X , y nos gustaría encontrar una distribución de probabilidad P_Y , con función de densidad f_Y , tal que P_Y sea una buena aproximación de P_X . Por tanto, dos conjuntos de datos son similares si son ejemplos de dos distribuciones de probabilidad aproximadas (o de una misma distribución de probabilidad).

Después de haber matematizado nuestro objetivo, podemos preguntarnos por qué no definimos $P_X = P_Y$ y tomamos muestras de P_X para generar objetos similares.

El problema es que desconocemos P_X y solo conocemos un conjunto finito de muestras X . Por tanto, nuestro estudio se centrará en aprender la distribución P_X solo a partir de su conjunto finito de muestras y luego encontrar P_Y como proceso de aproximación de P_X .

2.3.1. Enfoque básico de las Redes Generativas Adversarias (GANs)

Inicialmente, se comienza con una distribución de probabilidad inicial P_Z definida en $\mathbb{R}^d$ con objeto de conseguir aproximar P_X . Establecemos que P_Z sea la distribución normal $N(0, I_d)$, la variable aleatoria Z sigue una distribución normal, aunque podría haber sido cualquier otra distribución.

El enfoque básico de las GANs es encontrar una función $G : \mathbb{R}^d \rightarrow \mathbb{R}^n$ tal que si una variable aleatoria $z \in \mathbb{R}^d$ tiene una distribución de probabilidad P_Z entonces la variable $G(z)$ debe tener una distribución P_X . Claramente, la distribución de $G(z)$ es $P_Z \circ G^{-1}$. Por tanto, queremos encontrar una función $G(z)$ tal que $P_Z \circ G^{-1} = P_X$ ó $P_Z \circ G^{-1}$ ser una buena aproximación de P_X .

Partiendo de que solo conocemos muestras de P_X , si llegamos a determinar G tendremos muestras $G(z)$ donde z se extrae de la distribución P_Z . Pero aparece el problema de como determinar a partir de estas muestras que nuestra distribución $P_Z \circ G^{-1}$ es la misma o una buena aproximación de P_X .

La técnica de las GANs es introducir una función discriminadora $D(x)$ que intenta descartar las muestras generadas por G y decir que son muestras falsas. $D(x)$ es un clasificador que trata de distinguir las muestras del conjunto de entrenamiento X de las muestras generadas $G(z)$. Al principio la función discriminadora D sabe perfectamente distinguir las muestras pero más adelante cuando G aprenda a generar muestras más similares a X , G estará aprendiendo y le tocará al discriminador mejorar para ir distinguiendo correctamente. A través de este proceso de entrenamiento se alcanza un punto de equilibrio donde las muestras generadas deberán tener una distribución muy similar a las muestras de entrenamiento X .

Para relacionar todo el aspecto matemático con el informático nos preguntamos donde aparecen las redes neuronales y el aprendizaje profundo en todo esto. La respuesta nos la dan los teoremas de Aproximación Universal, ya que afirman que las redes neuronales profundas se pueden usar para aproximar casi cualquier función, mediante el aprendizaje de los parámetros de la red utilizando los conjuntos de entrenamiento.

En definitiva, modelaremos tanto la función discriminadora D como la función generadora G como redes neuronales con parámetros ω y θ . Por tanto, escribiremos $D(x)$ como $D_\omega(x)$ y $G(z)$ como $G_\theta(z)$ y denotaremos $P_{y_\theta} := P_Z \circ G_\theta^{-1}$. Nuestro objetivo es encontrar la función $G_\theta(z)$ ajustando el valor de θ mediante el entrenamiento de la red.

2.3.2. Problema min-max

Vamos a estudiar una serie de definiciones y teoremas en $\mathbb{R}^2$ pero pueden extraerse a $\mathbb{R}^n$.

Definición 2.1. Un punto de silla $(\bar{x}, \bar{y})$ de una función g es un punto crítico que no es extremo local de la función. Esto quiere decir que para cualquier entorno del punto $(\bar{x}, \bar{y})$ existen puntos (x_1, y_1) y (x_2, y_2) del entorno verificando

$$g(x_1, y_1) \leq g(\bar{x}, \bar{y}) \leq g(x_2, y_2)$$

Definición 2.2. Un punto minmax de una función $g : \mathbb{R}^2 \rightarrow \mathbb{R}$ es un punto $(\bar{x}, \bar{y}) \in \mathbb{R}^2$ que verifica la siguiente igualdad:

$$g(\bar{x}, \bar{y}) = \min_x \max_y g(x, y) = \max_y \min_x g(x, y)$$

Teorema 2.1. Si una función $g : \mathbb{R}^2 \rightarrow \mathbb{R}$ tiene un punto minmax $(\bar{x}, \bar{y}) \in \mathbb{R}^2$, esto es

$$g(\bar{x}, \bar{y}) = \min_x \max_y g(x, y) = \max_y \min_x g(x, y)$$

entonces la función g tiene un punto de silla con respecto a $\mathbb{R}^2$ y $(\bar{x}, \bar{y})$ es el punto de silla.

Demostración. De la hipótesis del teorema tenemos

$$\min_x g(x, \bar{y}) = g(\bar{x}, \bar{y}) = \max_y g(\bar{x}, y)$$

Consecuentemente, para cualquier $(x, y) \in \mathbb{R}^2$ que se encuentre en un entorno local de $(\bar{x}, \bar{y})$ tenemos

$$g(x, \bar{y}) \geq g(\bar{x}, \bar{y}) \geq g(\bar{x}, y)$$

Veamos que $(\bar{x}, \bar{y})$ es un punto de crítico de g . Por un lado, tenemos que el punto $(\bar{x}, \bar{y})$ alcanza el mínimo local en g con respecto a la variable x , esto es, $\min_x g(x, \bar{y}) = g(\bar{x}, \bar{y})$. Entonces, tenemos que $\frac{\partial g}{\partial x}(\bar{x}, \bar{y}) = 0$. Por otro lado, como el punto $(\bar{x}, \bar{y})$ es un máximo local de la función g con respecto a la variable y , entonces $\frac{\partial g}{\partial y}(\bar{x}, \bar{y}) = 0$. Finalmente, se obtiene que es un punto crítico, ya que

$$\nabla g(\bar{x}, \bar{y}) = \left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y} \right)(\bar{x}, \bar{y}) = (0, 0)$$

lo que implica que $(\bar{x}, \bar{y})$ es un punto de silla. $\square$

2.3.3. Ejemplo de problema min-max

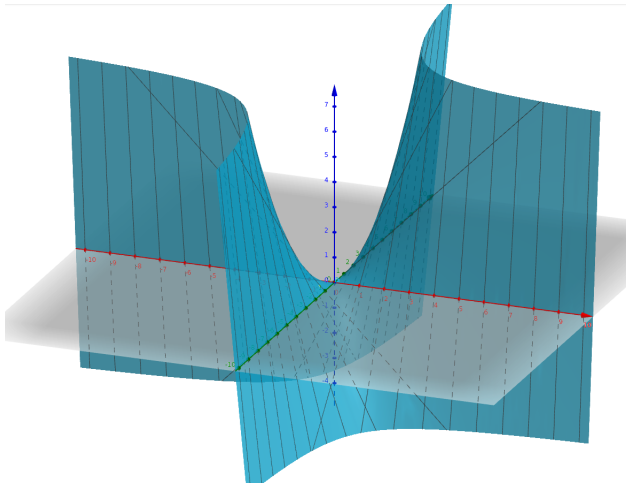


Figura 2.3.: Ejemplo MinMax f

Para determinar que el punto $(0, 0)$ es un punto de silla podemos usar Teorema 2.1. Veamos que el punto $(0, 0)$ es un punto minmax, esto es, que cumple

$$f(0, 0) = \min_x \max_y f(x, y) = \max_y \min_x f(x, y)$$

Para obtener $\min_x f(x, y)$ calculamos $\frac{\partial f(x, y)}{\partial x} = y$, tenemos que $\frac{\partial f(x, y)}{\partial x} = 0$, si y solo si, $y = 0$. Asimismo, para $\max_y f(x, y)$ calculamos $\frac{\partial f(x, y)}{\partial y} = x$, tenemos que $\frac{\partial f(x, y)}{\partial y} = 0$, si y solo si, $x = 0$. Finalmente, el punto minmax es $(0, 0)$ y aplicando el teorema tenemos que es el punto de silla de f .

2.3.4. Formulación min-max del enfoque básico de las GANs

El enfoque básico de las GANs descrito en la Subsección 2.3.1, conocido en inglés como Vanilla GAN, se puede formular mediante un problema min-max entre dos funciones: el generador $G : \mathbb{R}^d \rightarrow \mathbb{R}^n$ y el discriminador $D : \mathbb{R}^n \rightarrow [0, 1]$.

Vamos a usar la siguiente función de pérdida como función objetivo del problema min-max:

$$V(D, G) = \mathbb{E}_{x \sim P_{\mathcal{X}}}[\log D(x)] + \mathbb{E}_{z \sim P_{\mathcal{Z}}}[\log(1 - D(G(z)))] \quad (2.1)$$

donde $\mathbb{E}$ denota la esperanza con respecto a la distribución de probabilidad especificada en el subíndice (La notación $X \sim P_{\mathcal{X}}$ significa que los datos X son generados a partir de la distribución de probabilidad $P_{\mathcal{X}}$). Vamos a resolver el siguiente problema min-max:

$$\min_G \max_D V(D, G) = \min_G \max_D (\mathbb{E}_{x \sim P_{\mathcal{X}}}[\log D(x)] + \mathbb{E}_{z \sim P_{\mathcal{Z}}}[\log(1 - D(G(z)))])$$

- Dado un generador G , $\max_D V(D, G)$ obtiene el discriminador D que rechaza las muestras generadoras $G(z)$ ya que intenta asignar valores altos a las muestras de la distribución $P_{\mathcal{X}}$ y valores bajos a las muestras generadas $G(z)$.

Básicamente, al maximizar estamos asignándole un valor alto a $D(x)$ cuando x es una muestra de la distribución $P_{\mathcal{X}}$ y asignándole un valor bajo a $D(G(z))$ cuando z es una muestra de $P_{\mathcal{Z}}$. Esto equivale a decir que el discriminador le asigne una probabilidad alta a las muestras de entrenamiento y una probabilidad baja a las muestras falsas construidas a partir del generador.

- Dado un discriminador D , $\min_G V(D, G)$ obtiene el generador G que genera muestras $G(z)$ que intentan “engañar” al discriminador D para que le asigne valores altos.

Analizándolo con detalle, nos damos cuenta de que para alcanzar el mínimo le tenemos que asignar un valor alto a $D(G(z))$ cuando z es una muestra de $P_{\mathcal{Z}}$, así $(1 - D(G(z)))$ alcanza el mínimo, y, por otra parte, el valor $D(x)$ es constante, ya que viene dado por el discriminador. Es decir, el discriminador le asigna una probabilidad alta a las muestras falsas construidas a partir del generador, de esta forma se consigue engañar al discriminador.

Sea $y = G(z)$ que tiene distribución de probabilidad $P_Y := P_{\mathcal{Z}} \circ G^{-1}$, podemos reescribir $V(D, G)$ en términos de D y P_Y como

$$\begin{aligned} \bar{V}(D, P_Y) &:= V(D, G) = \mathbb{E}_{x \sim P_{\mathcal{X}}}[\log D(x)] + \mathbb{E}_{z \sim P_{\mathcal{Z}}}[\log(1 - D(G(z)))] \\ &= \mathbb{E}_{x \sim P_{\mathcal{X}}}[\log D(x)] + \mathbb{E}_{y \sim P_Y}[\log(1 - D(y))] \\ &= \int_{\mathbb{R}^n} \log D(x) dP_{\mathcal{X}}(x) + \int_{\mathbb{R}^n} \log(1 - D(y)) dP_Y(y) \end{aligned}$$

El problema minmax estudia

$$\min_G \max_D V(D, G) = \min_G \max_D \left(\int_{\mathbb{R}^n} \log D(x) dP_{\mathcal{X}}(x) + \int_{\mathbb{R}^n} \log(1 - D(y)) dP_{\mathcal{Y}}(y) \right)$$

Supongamos que la distribución de probabilidad $P_{\mathcal{X}}$ tiene a $f_{\mathcal{X}}$ como función de densidad y la distribución $P_{\mathcal{Y}}$ a $f_{\mathcal{Y}}$ como función de densidad. Entonces

$$\bar{V}(D, P_{\mathcal{Y}}) = \int_{\mathbb{R}^n} (\log D(x) f_{\mathcal{X}}(x) + \log(1 - D(x)) f_{\mathcal{Y}}(x)) dx$$

Haciendo uso de las funciones de densidad, el problema minmax se puede reescribir como

$$\min_G \max_D V(D, G) = \min_G \max_D \int_{\mathbb{R}^n} (\log D(x) f_{\mathcal{X}}(x) + \log(1 - D(x)) f_{\mathcal{Y}}(x)) dx$$

Si usamos $\bar{V}(D, P_{\mathcal{Y}})$, entonces nuestro problema minmax sería $\min_{P_{\mathcal{Y}}} \max_D \bar{V}(D, P_{\mathcal{Y}})$ bajo la restricción $P_{\mathcal{Y}} = P_{\mathcal{Z}} \circ G^{-1}$. Nos vamos a centrar en el caso de que conocemos las funciones de densidad y en este caso se han establecido los siguientes resultados.

Proposición 1. *Dadas dos distribuciones de probabilidad continuas $P_{\mathcal{X}}$ y $P_{\mathcal{Y}}$ en $\mathbb{R}^n$ con funciones de densidad $f_{\mathcal{X}}$ y $f_{\mathcal{Y}}$ respectivamente,*

$$\max_D \bar{V}(D, P_{\mathcal{Y}}) = \max_D \int_{\mathbb{R}^n} (\log D(x) f_{\mathcal{X}}(x) + \log(1 - D(x)) f_{\mathcal{Y}}(x)) dx$$

es alcanzado por $D_{f_{\mathcal{X}}, f_{\mathcal{Y}}}(x) = \frac{f_{\mathcal{X}}(x)}{f_{\mathcal{X}}(x) + f_{\mathcal{Y}}(x)}$ para x en el interior de $\text{supp}(P_{\mathcal{X}}) \cup \text{supp}(P_{\mathcal{Y}})$, donde $\text{supp}(P_{\mathcal{X}})$ es la clausura del conjunto $\{x \in \mathbb{R}^n | P_{\mathcal{X}}(x) \neq 0\}$.

2.3.5. Medidas de similitud entre dos distribuciones de probabilidad

Vamos a estudiar formas de medir la distancia entre dos distribuciones de probabilidad continuas $P_{\mathcal{X}}$ y $P_{\mathcal{Y}}$. Primero, vamos a presentar el concepto de f -divergencia junto con sus propiedades y luego detallaremos casos particulares como son las divergencias de Kullback-Leibler(KL) y de Jensen-Shannon(JSD). Además, probaremos un resultado de f -divergencia que necesitaremos para demostrar el teorema que nos proporciona la solución del enfoque básico de las GANs formulado como un problema minmax.

El método que utiliza las GANs para medir la diferencia o la distancia entre la distribución generada y la distribución real es la divergencia de Jensen-Shannon que se construye a partir de la divergencia de Kullback-Leibler.

La divergencia KL vamos a ver que mide la diferencia entre dos distribuciones pero no es realmente una distancia porque no es simétrica, para medir la distancia entre las distribuciones de probabilidad usaremos la divergencia JSD que si es simétrica.

f-divergencia

Definición 2.3. Sea f una función estrictamente convexa con dominio $I \subseteq \mathbb{R}$ tal que $f(1) = 0$. Suponemos que $f(x) = +\infty$ para todo $x \notin I$. Sean $f_{\mathcal{X}}(x)$ y $f_{\mathcal{Y}}(x)$ dos funciones de densidad de probabilidad en $\mathbb{R}^n$. Entonces la f -divergencia de $f_{\mathcal{X}}$ y $f_{\mathcal{Y}}$ es definida como

$$D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) := \mathbb{E}_{x \sim P_{\mathcal{Y}}}[f(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)})] = \int_{\mathbb{R}^n} f(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)}) f_{\mathcal{Y}}(x) dx$$

donde nosotros adoptamos por convención que $f(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)}) f_{\mathcal{Y}}(x) = 0$ si $f_{\mathcal{Y}}(x) = 0$

Proposición 2. Sea $f(x)$ una función estrictamente convexa en el dominio $I \subseteq \mathbb{R}$ tal que $f(1) = 0$. Supongamos que $\text{supp}(f_{\mathcal{X}}) \subseteq \text{supp}(f_{\mathcal{Y}})$ (es equivalente a decir $f_{\mathcal{X}} \ll f_{\mathcal{Y}}$) o que $f(t) > 0$ para $t \in [0, 1]$. Entonces, $D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) \geq 0$, y $D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) = 0$ si y solo si $f_{\mathcal{X}} = f_{\mathcal{Y}}$.

Observación 1. La proposición anterior nos afirma que la f -divergencia se comporta como una distancia, aunque puede no ser simétrica.

Casos particulares

- Consideramos la f -divergencia con $f(x) = -\log(x)$ y vamos a ver que cumple las condiciones de Definición 2.3. Sabemos que la función logaritmo es estrictamente cóncava, entonces f es estrictamente convexa. Además, tenemos que $f(1) = -\log(1) = 0$. Por tanto, tenemos la siguiente f -divergencia

$$\begin{aligned} D_f(f_{\mathcal{Y}}||f_{\mathcal{X}}) &:= \mathbb{E}_{x \sim P_{\mathcal{X}}}[f(\frac{f_{\mathcal{Y}}(x)}{f_{\mathcal{X}}(x)})] = \int_{\mathbb{R}^n} f(\frac{f_{\mathcal{Y}}(x)}{f_{\mathcal{X}}(x)}) f_{\mathcal{X}}(x) dx \\ &= \int_{\mathbb{R}^n} -\log(\frac{f_{\mathcal{Y}}(x)}{f_{\mathcal{X}}(x)}) f_{\mathcal{X}}(x) dx = \int_{\mathbb{R}^n} \log(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)}) f_{\mathcal{X}}(x) dx \end{aligned}$$

Definición 2.4. La divergencia de Kullback-Leibler mide la distancia entre dos funciones de densidad, $f_{\mathcal{X}}$ y $f_{\mathcal{Y}}$, también se conoce como divergencia de información y entropía relativa, viene dada por:

$$KL(f_{\mathcal{X}}||f_{\mathcal{Y}}) = \int_{\mathbb{R}^n} f_{\mathcal{X}}(x) \log(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)}) dx$$

Observación 2. La divergencia de Kullback-Leibler $KL(f_{\mathcal{X}}||f_{\mathcal{Y}})$ es finita si y solo si $f_{\mathcal{Y}}(x) \neq 0$ en $\text{supp}(f_{\mathcal{X}})$ (ó equivalentemente $\text{supp}(f_{\mathcal{X}}) \subset \text{supp}(f_{\mathcal{Y}})$) Notemos que si $f_{\mathcal{X}}(x) \neq 0$ entonces tenemos $f_{\mathcal{Y}}(x) \neq 0$.

Lema 1. La divergencia de Kullback-Leibler entre dos funciones de densidad, $f_{\mathcal{X}}$ y $f_{\mathcal{Y}}$, cumple

1. $KL(f_{\mathcal{X}}||f_{\mathcal{Y}}) \geq 0$
2. No es simétrica.

- Consideramos la f -divergencia $f : (0, +\infty) \rightarrow \mathbb{R}$ con $f(x) = \frac{1}{2}x \log(x) - \frac{1}{2}(x+1) \log(\frac{x+1}{2})$ y vamos a ver que cumple las condiciones de Definición 2.3. Primero tenemos que probar que f es estrictamente convexa, para ello estudiamos el signo de la segunda derivada. Sabemos que si la segunda derivada de f es positiva, entonces la función f es estrictamente convexa,

$$f''(x) = \frac{1}{2x^2+2x} > 0 \quad \forall x \in (0, +\infty)$$

Además, tenemos que $f(1) = \frac{1}{2} \log(1) - \frac{1}{2}(1+1) \log(\frac{1+1}{2}) = 0 + 0 = 0$. Por tanto, tenemos la siguiente f -divergencia, que es justo una simetrización de la divergencia anterior.

$$\begin{aligned} D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) &= \int_{\mathbb{R}^n} f\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) f_{\mathcal{Y}} \\ &= \int_{\mathbb{R}^n} \frac{1}{2} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \log\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) - \frac{1}{2} \left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1\right) \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \log\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) - \left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1\right) \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \left(\log\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) - \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right)\right) - \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \left(\log\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) - \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right)\right) + \log\left(\frac{2}{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \log\left(\frac{2f_{\mathcal{X}}}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) + \log\left(\frac{2f_{\mathcal{Y}}}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} f_{\mathcal{X}} \log\left(\frac{2f_{\mathcal{X}}}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) + f_{\mathcal{Y}} \log\left(\frac{2f_{\mathcal{Y}}}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) \\ &= \frac{1}{2} (KL(f_{\mathcal{X}}||\frac{f_{\mathcal{X}} + f_{\mathcal{Y}}}{2}) + KL(f_{\mathcal{Y}}||\frac{f_{\mathcal{X}} + f_{\mathcal{Y}}}{2})) \end{aligned}$$

Definición 2.5. La divergencia de Jensen-Shannon es otro método para medir la distancia entre dos distribuciones de probabilidad, es una versión simetrizada y suavizada de la divergencia de Kullback-Leibler $KL(f_{\mathcal{X}}||f_{\mathcal{Y}})$ definida como

$$JSD(f_{\mathcal{X}}||f_{\mathcal{Y}}) = \frac{1}{2} (KL(f_{\mathcal{X}}||\frac{f_{\mathcal{X}} + f_{\mathcal{Y}}}{2}) + KL(f_{\mathcal{Y}}||\frac{f_{\mathcal{X}} + f_{\mathcal{Y}}}{2}))$$

Lema 2. La divergencia de Jensen-Shannon entre dos funciones de densidad, $f_{\mathcal{X}}$ y $f_{\mathcal{Y}}$, cumple

1. $JSD(f_{\mathcal{X}}||f_{\mathcal{Y}}) \geq 0$
2. Es simétrica.

2.3.6. Solución al problema minmax

Bajo la suposición de que las distribuciones de probabilidad son continuas, tenemos el siguiente resultado que nos proporciona la solución al problema minmax planteado en nuestro enfoque básico de las GANs. Obsérvese que si las distribuciones de probabilidad no fuesen continuas, no tendríamos la solución de nuestro problema. Más adelante, generalizaremos este teorema para tener un resultado que nos dé la solución para cualesquiera distribuciones de probabilidad P_X y P_Y .

Teorema 2.2. *Sea $f_X(x)$ una función de densidad de probabilidad en $\mathbb{R}^n$. Para la distribución de probabilidad continua P_Y con función de densidad $f_Y(x)$ y $D : \mathbb{R}^n \rightarrow [0, 1]$ consideramos el problema minmax*

$$\min_{P_Y} \max_D \bar{V}(D, P_Y) = \min_{P_Y} \max_D \int_{\mathbb{R}^n} (\log D(x)f_X(x) + \log(1 - D(x))f_Y(x))dx$$

Entonces la solución se obtiene con $f_Y = f_X$ y $D(x) = 1/2$ para todo $x \in \text{supp}(P_X)$.

2.3.7. Algoritmo de entrenamiento de las Redes Generativas Adversarias (GANs)

En esta sección se va a presentar el algoritmo de entrenamiento del enfoque básico de las GANs que trata de optimizar la ecuación matemática presentada anteriormente, $V(D, G)$, usando el descenso de gradiente estocástico (SGD). Nuestro objetivo es encontrar los pesos de los modelos generador y discriminador que optimizan nuestra función. Primero, se presentará el algoritmo y a continuación, enunciaremos la convergencia de dicho algoritmo.

Hay que tener en cuenta que el problema minmax sin restricciones no es el mismo problema que el problema minmax original donde P_Y se restringe a $P_Y = P_Z \circ G^{-1}$ (Ecuación 2.3.4). En la práctica ambos presentarán las mismas propiedades, de hecho restringiremos las funciones del generador y del discriminador para que actúen como redes neuronales presentando dicha formulación las mismas propiedades. En consecuencia, tenemos las funciones: $D = D_\omega$ y $G = G_\theta$ donde ω son los pesos que aprende la red discriminadora y θ los pesos que aprende la red generadora, y ajustando $P_{Y_\theta} = P_Z \circ G_\theta^{-1}$ nuestro problema minmax viene dado como

$$\begin{aligned} \min_{\theta} \max_{\omega} V(D_\omega, G_\theta) &= \min_{\theta} \max_{\omega} \mathbb{E}_{x \sim P_X} [\log D_\omega(x)] + \mathbb{E}_{z \sim P_Z} [\log(1 - D_\omega(G_\theta(z)))] \\ &= \min_{\theta} \max_{\omega} \int_{\mathbb{R}^n} (\log D_\omega(x)dP_X + \log(1 - D_\omega(G_\theta(z)))dP_{Y_\theta}) \end{aligned}$$

Obsérvese que cada vez que aparezca D_ω y G_θ estamos nombrando a la red neuronal que modela el discriminador y el generador.

Algorithm 12: Algoritmo entrenamiento GANs

■ Para un número de iteraciones de entrenamiento **hacer**

• Para k pasos **hacer**

- Tomar una muestra de m ejemplos $\{z_1, \dots, z_m\}$ en $\mathbb{R}^n$ de la distribución P_Z .
- Tomar una muestra de m ejemplos $\{x_1, \dots, x_m\} \subset X$ del conjunto de entrenamiento X .
- Actualizar el discriminador D_ω por ascenso de su gradiente estocástico con respecto a ω :

$$\nabla_\omega \frac{1}{m} \sum_{i=1}^m [\log D_\omega(x_i) + \log(1 - D_\omega(G_\theta(z_i)))]$$

• **fin**

- Tomar una muestra de m ejemplos $\{z_1, \dots, z_m\}$ en $\mathbb{R}^n$ de la distribución P_Z .
- Actualizar el generador G_θ por descenso de su gradiente estocástico con respecto a θ :

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^m [\log(1 - D_\omega(G_\theta(z_i)))]$$

■ **fin**

El siguiente resultado nos sirve para afirmar la convergencia en el algoritmo de optimización iterativo y la existencia de un punto de equilibrio. Una iteración del bucle de este algoritmo consiste en actualizar primero D para un f_Y dado y luego actualizar f_Y con el nuevo D . Repetir este proceso nos conducirá a la solución deseada.

Proposición 3. *Si en cada ciclo, el discriminador D se le permite alcanzar su óptimo dado $f_Y(x)$, seguido de una actualización de f_Y para mejorar el criterio de minimización*

$$\min_{f_Y} \int_{\mathbb{R}^n} (\log D(x) f_X(x) + \log(1 - D(x)) f_Y(x)) dx$$

entonces f_Y converge a f_X .

Una vez probada la convergencia del algoritmo, se dieron cuenta de que al actualizar la red neuronal G_θ a partir de minimizar $\mathbb{E}[\log(1 - D_\omega(G_\theta(z)))]$ puede saturarse antes de que finalice el algoritmo. Por tanto, se decidió minimizar $-\mathbb{E}[\log D_\omega(G_\theta(z))]$ en lugar de $\mathbb{E}[\log(1 - D_\omega(G_\theta(z)))]$.

3. Aprendizaje por Imitación

En este capítulo, se introduce el paradigma de RL conocido como aprendizaje por imitación, estudiando sus fundamentos y esquematizando sus distintos enfoques. A continuación, centramos nuestra atención en profundizar nuestra investigación en el aprendizaje por imitación generativo adversario (GAIL). Los siguientes estudios se han analizados en profundidad [HE16], [AN04], [Sew19].

3.1. Fundamentos

La tarea dentro del aprendizaje por refuerzo donde se aprende a partir de un agente experto sin acceso al entorno, ni a la señal de recompensa, se conoce como **aprendizaje por imitación** (*Imitation Learning*), también conocida como aprendizaje por aprendizaje (*Apprenticeship Learning*). El agente hace uso de información adicional para aprender del agente experto, concretamente de demostraciones expertas. Estas demostraciones, contienen las elecciones de la política experta, la política que utiliza conocimiento experto y por tanto, la política que seguirá el agente experto. Entonces, la política de nuestro agente imitará a la política experta, más bien a las acciones que seleccionará cuando tengamos una determinada observación del entorno.

Su objetivo es encontrar una política que sea similar a la del experto. Para lograr esto, utiliza una función de costo que mide la diferencia entre las acciones tomadas por la política del agente y las acciones tomadas por la política del experto para un mismo estado. A continuación, se ajusta la política del agente para minimizar esta función de costo y hacerla más similar a la del experto.

Otros enfoques de aprendizaje por refuerzo lo que requieren es que el agente aprenda por prueba y error, mientras que en este marco lo que se pretende es que el agente aprenda observando ejemplos del comportamiento del experto y tratando de imitarlo.

Este aprendizaje tiene muchas aplicaciones, como la conducción autónoma, la robótica y los videojuegos. Es especialmente útil cuando la tarea es difícil de especificar mediante una función de recompensa y cuando el comportamiento del agente es conocido y puede ser utilizado como guía para el aprendizaje del agente.

En 2004, Abbel y Ng [AN04], definieron el concepto de aprendizaje por imitación con el formalismo de aprendizaje por aprendizaje: el agente inteligente tiene que encontrar una política π que funcione al menos tan bien como la política experta π_E respecto a una función de recompensa desconocida $r(s, a)$. Para ello, debemos de medir cómo de buena es una política y lo haremos con la medida de ocupación (sección 3.2.3).

Ahora bien, un problema de aprendizaje por imitación se convierte en un problema de coincidencia de medidas donde lo que nos interesa es que la medida de la política

experta coincide con la medida de la política de nuestro agente inteligente. El objetivo del aprendizaje por imitación es aprender la siguiente política π , donde ψ^* es una medida de distancia entre ρ_π y ρ_{π_E} , y $H(\pi)$ es un término de normalización con factor de compensación λ , definido como $H(\pi) \triangleq \mathbb{E}_\pi[-\log \pi(s, a)]$.

$$\pi = \arg \min_{\pi \in \Pi} \psi^*(\rho_\pi - \rho_{\phi_E}) - \lambda H(\pi)$$

Existen diferentes enfoques para aprender una política a partir de demostraciones expertas:

1. Clonación del comportamiento (*Behavioral Cloning, CL*): es la forma mas simple ya que utiliza las demostraciones expertas en un enfoque de aprendizaje supervisado y a partir de este enfoque se construyen métodos mas complejos. Se aplica el enfoque de aprendizaje supervisado, considerando las observaciones como las características y las acciones como las etiquetas.
2. Aprendizaje por refuerzo inverso (*Inverse Reinforcement Learning, IRL*): es muy útil en las aplicaciones donde puede ser complicado especificar la función de recompensa explícita. Este enfoque nos permite recuperar la función de recompensa que desconocemos a partir de las demostraciones expertas y utilizarla en un proceso de aprendizaje donde se trate de imitar al experto.
3. Aprendizaje por imitación de observaciones (*Ifo, ILFO*): resuelve las limitaciones del aprendizaje por imitación que generalmente requiere acciones como etiquetas para las entradas del estado. Es algo que ocurre con frecuencia en el proceso de aprendizaje por imitación en humanos.
4. Métodos probabilísticos - Procesos Gaussianos: Los métodos desde una perspectiva de inferencia probabilística incluyen el uso de regresión de procesos gaussianos para representar los datos de demostración y, por lo tanto, guiar la política de acción, lo cual es una alternativa más eficiente en algunos casos a los métodos de redes neuronales profundas.
5. Otros enfoques: Un ejemplo sería incorporar demostraciones en un búfer de reproducción para el aprendizaje por refuerzo off-policy.

Finalmente, podemos visualizar un esquema del aprendizaje por imitación donde podemos apreciar la ubicación de la técnica GAIL que procederemos a estudiar en detalle.

3.2. Aprendizaje por Imitación Generativo Adversario (GAIL)

3.2.1. Concepto

El aprendizaje por refuerzo y las redes generativas adversarias son dos conceptos que GAIL [HE16] trata de conectar.

GAIL pretende aprender una política a partir del comportamiento de un experto, sin interacción con el experto ni acceso a la recompensa.

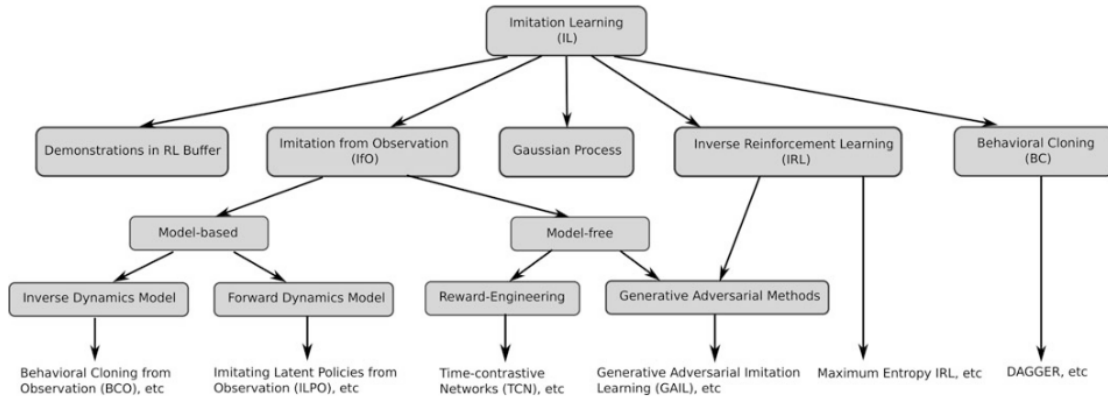


Figura 3.1.: Algoritmos de Aprendizaje por Imitación[Sew19]

Inicialmente, podemos pensar que una buena idea sería recuperar la función de costo del experto con aprendizaje por refuerzo inverso (IRL) y después extraer una política de esa función de costo con aprendizaje por refuerzo (RL). El aprendizaje por refuerzo inverso trata de aprender una función de costo, que explica el comportamiento del agente, pero no le dice como actuar al agente.

Este enfoque de RL+IRL, esto es, aplicar primero IRL y después RL, es indirecto, muy costoso y puede ser lento. Muchos algoritmos de aprendizaje por refuerzo inverso son extremadamente costosos. Nos cuestionamos, entonces, ¿por qué debemos aprender una función de costo si al hacerlo caeremos en un gasto computacional significativo y no produciremos acciones directamente?

Deseamos un algoritmo que nos diga cómo debe actuar el Agente aprendiendo directamente de la política. Esto se consigue con el algoritmo de Aprendizaje por Imitación model-free. Así, nace el marco de GAIL, donde se propone extraer la política directamente de los datos. Este marco tiene la gran ventaja de que establece una analogía entre el aprendizaje por imitación y las redes generativas adversarias (GANs), vistas en el capítulo 2.

Recordamos que nuestro problema básico a estudiar es el aprendizaje por imitación: estudiar cómo aprender a realizar acciones a partir de demostraciones de acciones expertas o acciones consideradas buenas para un estado determinado.

Visualizando el esquema 3.1 vemos como GAIL pertenece a los métodos model-free IfO y además, pertenece a los métodos generativos adversariales, por tanto, podemos usarlo perfectamente para establecer la conexión que deseamos.

Trabajaremos con un espacio de estados finitos $\mathcal{S}$ y con un espacio de acciones finitas $\mathcal{A}$. $\Pi = \{\pi : \mathcal{S} \rightarrow \mathcal{A}\}$ es el conjunto de políticas estocásticas estacionarias que nos dan acciones de $a \in \mathcal{A}$ dado estados en $s \in \mathcal{S}$. Los estados sucesores se extraen de un modelo dinámico $P(s'|s, a)$.

Utilizaremos una esperanza con respecto a la política $\pi \in \Pi$ que se denota:

$$\mathbb{E}_\pi[c(s, a)] \triangleq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)\right] \quad (3.1)$$

donde $s_0 \sim p_0$, $a_t \sim \pi(\cdot|s_t)$ y $s_{t+1} \sim \pi(\cdot|s_t, a_t) \forall t \geq 0$

Observación 3. Como estamos trabajando con aprendizaje por refuerzo, entonces una función de costo $c \in C$ evaluada en un estado $s \in \mathcal{S}$ y una acción $a \in \mathcal{A}$, se corresponde con la recompensa obtenida cuando el agente ejecuta la política actual con la asignación de acción a a la observación s del entorno, $\pi(s) = a$.

3.2.2. Aprendizaje por Refuerzo Inverso (IRL)

Sea π_E la política experta, $\Pi = \{\pi : \mathcal{S} \rightarrow \mathcal{A}\}$ el conjunto donde se almacenan todas las políticas, $\mathcal{C} := \mathbb{R}^{\mathcal{S} \times \mathcal{A}} = \{c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}\}$ el conjunto de todas las funciones de costo que podemos definir para las secuencias estado-acción, tenemos el problema de optimización donde tenemos que maximizar en $c \in \mathcal{C}$ la siguiente función:

$$\min_{\pi \in \Pi} (-H(\pi) + \mathbb{E}_\pi[c(s, a)]) - \mathbb{E}_{\pi_E}[c(s, a)] \quad (3.2)$$

donde $H(\pi) \triangleq \mathbb{E}_\pi[-\log \pi(a|s)]$.

Para enlazar aprendizaje por refuerzo y aprendizaje por refuerzo inverso, procedemos a mostrar ambos conceptos.

- **Aprendizaje por refuerzo (RL):** para cada función de coste c , **obtiene la política** π que minimiza $RL(c)$

$$c \in \mathcal{C} := \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \xrightarrow{RL} \pi \in \Pi$$

$$RL(c) = \arg \min_{\pi \in \Pi} (-H(\pi) + \mathbb{E}_\pi[c(s, a)]) \quad (3.3)$$

- **Aprendizaje por refuerzo inverso (IRL) 2 :**

$\forall \pi \in \Pi$, el aprendizaje por refuerzo inverso calcula el valor $IRL_\psi(\pi_e)$ regularizado por ψ . Esto es, la **función de costo** c que maximiza la diferencia entre la Ecuación 3.2.2 y $\psi(c)$:

$$\pi_E \in \Pi \xrightarrow{IRL_\psi} c \in \mathcal{C} := \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$$

$$IRL_\psi(\pi_E) = \arg \max_{c \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}} -\psi(c) + (\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_\pi[c(s, a)]) - \mathbb{E}_{\pi_E}[c(s, a)] \quad (3.4)$$

Anteriormente, se ha comentado que obtener una función de costo, sin tener una política asociada a dicha función de costo, no tiene mucho sentido, ya que lo que deseamos es conocer las acciones que debe realizar nuestro agente para un entorno específico. Por tanto, en nuestro estudio si calculamos la función de costo $\bar{c} \in IRL_\psi$, entonces calcularemos la política dada por $RL(\bar{c}) \in \Pi$ ya que de ella podremos extraer las acciones que ejecutará nuestro agente en su entorno.

$$\pi_E \in \Pi \xrightarrow{IRL_\psi} c \in \mathcal{C} \xrightarrow{RL} \pi \in \Pi$$

De esta forma, estamos calculando una política π tal que $\pi \in RL \circ IRL_\psi(\pi_E)$. Es decir, necesitamos un algoritmo que calcule la política $\pi \in \Pi$ verificando

$$\pi \in RL \circ IRL_\psi(\pi_E).$$

3.2.3. Medidas de ocupación

Sea $\tilde{c} \in IRL_\psi(\pi_E)$, estamos interesados en estudiar la política dada por $\pi \in RL(\tilde{c})$, esto es, la política π dada por la ejecución de aprendizaje por refuerzo, RL , en la salida de IRL, esto es, en $\tilde{c}$. A continuación, vamos a caracterizar las políticas dadas por $RL(c)$, esto es, las políticas del siguiente conjunto:

$$RL \circ IRL_\psi(\pi_E)$$

Definición 3.1. Sea $\pi \in \Pi$ una política, definimos su **medida de ocupación**, $\rho_\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ como $\rho_\pi(s, a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \pi)$

La medida de ocupación puede ser interpretada como la **distribución de probabilidad** de la pareja estado-acción que un agente encuentra cuando navega por el entorno con la política π . Entonces podemos escribir el conjunto de todas las medidas de ocupación de las políticas de Π , $\mathcal{D} \triangleq \{\rho_\pi : \pi \in \Pi\}$, como un conjunto factible de restricciones afines: si $p_0(s)$ es la distribución de estados iniciales y $P(s'|s, a)$ es el modelo dinámico, entonces $\mathcal{D} = \{\rho : \rho \geq 0 \ \& \ \sum_a \rho(s, a) = p_0(s) + \gamma \sum_{s', a} P(s'|s, a) \rho(s', a), \forall s \in \mathcal{S}\}$. Además, hay una correspondencia uno a uno entre Π y $\mathcal{D}$.

Proposición 4. Si $\rho \in \mathcal{D}$, entonces ρ es la medida de ocupación para la política π_ρ definida como $\pi_\rho(a|s) \triangleq \rho(s, a) / \sum_{a'} \rho(s, a')$, y π_ρ es la única política cuya medida de ocupación es ρ .

Proposición 5.

$$RL \circ IRL_\psi(\pi_E) = \arg \min_{\pi \in \Pi} (-H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_E})) \quad (3.5)$$

Definición 3.2. Sea $f : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \rightarrow \bar{\mathbb{R}}$, entonces su **conjugada convexa** $f^* : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \rightarrow \bar{\mathbb{R}}$ viene dada como $f^*(x) = \sup_{y \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}} \{x^T y - f(y)\}$.

La Ecuación 3.5 nos está afirmando que el aprendizaje por refuerzo inverso ψ -regularizado, busca una política cuya medida de ocupación sea cercana a la del experto, medida por la función convexa ψ^* .

Básicamente, ψ^* , la conjugada convexa de ψ , mide cómo de distantes están las medidas de ocupación de la política experta y de la política que se desea aprender. En vista de ello, varias configuraciones de ψ conducirán a varios algoritmos de aprendizaje por imitación que resolverán directamente el problema de optimización planteado por la Ecuación 3.5.

Exploraremos tales algoritmos en las siguientes subsecciones, donde mostramos que ciertas configuraciones de ψ van a conducir tanto a algoritmos existentes como a uno novedoso.

Un caso especial es cuando ψ es una función constante, ya que es particularmente instructivo, por lo que establecemos y mostramos directamente los conceptos de optimización convexa.

Corolario 1. *Si ψ es una función constante, $\tilde{c} \in IRL_\psi(\pi_E)$, y $\tilde{\pi} \in RL(\tilde{c})$, entonces $\rho_{\tilde{\pi}} = \rho_{\pi_E}$*

De este argumento deducido de las proposiciones anteriores, se obtienen dos afirmaciones:

- **El aprendizaje por refuerzo (RL) es el dual de un problema de coincidencia de medida de ocupación** y la función de costo recuperada mediante IRL es el óptimo dual. Los algoritmos clásicos de IRL, como el algoritmo de Ziebart, que realizan iteraciones de aprendizaje por refuerzo en un bucle interno, pueden interpretarse como un proceso de ascenso dual. En este enfoque, se resuelve repetidamente el problema primal (aprendizaje por refuerzo) con valores duales fijos (costos). Sin embargo, es importante destacar que el ascenso dual solo es efectivo si la resolución del primal sin restricciones es eficiente. En el caso de la IRL, el ascenso dual es equivalente al aprendizaje por refuerzo en sí mismo.
- **La política óptima inducida es el óptimo primal.**
La política óptima inducida se obtiene ejecutando RL después de IRL, que es exactamente la operación de recuperar el óptimo primal a partir del óptimo dual; es decir, optimizando el Lagrangiano con las variables duales fijas en los valores óptimos duales. Si existe una dualidad fuerte, esto implica que la política óptima inducida es, de hecho, el óptimo primal, lo que significa que coincide con las medidas de ocupación del experto. Tradicionalmente, IRL se define como el proceso de encontrar una función de costo tal que la política del experto sea única y óptima. Sin embargo, ahora podemos ver alternativamente IRL como un procedimiento que *intenta inducir una política que coincida con la medida de ocupación del experto*.

Seguidamente, vamos a estudiar como realizar una selección práctica de medidas de ocupación. Hemos visto que si ψ es constante, tenemos que la medida de ocupación coincide con la medida del experto y este algoritmo no es útil en la práctica.

Definiendo $d_\psi(\rho_\pi, \rho_{\pi_E}) \triangleq \psi^*(\rho_\pi - \rho_{\pi_E})$, la Ecuación 3.5 pasa a resolver el siguiente problema:

$$\min_{\pi} d_\psi(\rho_\pi, \rho_{\pi_E}) - H(\pi) \quad (3.6)$$

3.2.4. Conexión Aprendizaje por Imitación y GANs

La conexión entre el aprendizaje por imitación y las redes generativas adversarias surge cuando utilizamos ciertas configuraciones de ψ .

Con estas configuraciones, la ecuación adquiere la forma de variantes regularizadas de algoritmos de aprendizaje por imitación existentes que se pueden escalar a entornos grandes con políticas parametrizadas. Más adelante, veremos como nuestra política se aproximará a partir de unos parámetros θ y estudiaremos π_θ .

Para una clase de funciones de costo $\mathcal{C} \subset \mathbb{R}^{S \times \mathcal{A}}$, un algoritmo de aprendizaje por imitación encuentra una política que funciona mejor que el experto en todo $\mathcal{C}$, optimizando el objetivo.

$$\min_{\pi} \max_{c \in \mathcal{C}} \mathbb{E}_{\pi}[c(s, a)] - \mathbb{E}_{\pi_E}[c(s, a)] \quad (3.7)$$

Demostraremos como la Ecuación 3.7 es un caso especial de Ecuación 3.6 para una cierta ψ .

RL seguido de IRL

Si tomamos ψ como la función indicadora $\delta_C : \mathbb{R}^{S \times \mathcal{A}} \rightarrow \bar{\mathbb{R}}$ definida como $\delta_C(c) = 0$ si $c \in \mathcal{C}$ y $+\infty$ en otro caso, podemos escribir el objetivo del aprendizaje por refuerzo inverso (IRL) como

$$\max_{c \in \mathcal{C}} \mathbb{E}_{\pi}[c(s, a)] - \mathbb{E}_{\pi_E}[c(s, a)] = \max_{c \in \mathbb{R}^{S \times \mathcal{A}}} -\delta_C(c) + \sum_{s, a} (\rho_{\pi}(s, a) - \rho_{\pi_E}(s, a)) c(s, a) = \delta_C^*(\rho_{\pi} - \rho_{\pi_E}) \quad (3.8)$$

Además, vemos que formulando el aprendizaje por refuerzo de la entropía regularizada:

$$\min_{\pi} -H(\pi) + \max_{c \in \mathcal{C}} \mathbb{E}_{\pi}[c(s, a)] - \mathbb{E}_{\pi_E}[c(s, a)] \quad (3.9)$$

es equivalente a actuar RL seguido de IRL con el coste del regularizador $\psi = \delta_C$, lo cual fuerza implícitamente el procedimiento IRL para recuperar una función de costo que se encuentra en $\mathcal{C}$. Anteriormente, se discutió que IRL presentaba ciertas limitaciones así que vamos a mostrar los inconvenientes en comparación a las ventajas de usar el procedimiento.

- Inconvenientes de IRL: Los algoritmos de aprendizaje por refuerzo inverso (IRL) no recuperan políticas similares a las de los expertos si $\mathcal{C}$ es demasiado restrictivo, como suele ser el caso de los subespacios lineales.
- Ventajas de IRL: Aunque las clases restrictivas pueden no llegar a una imitación exacta, el aprendizaje con dichas $\mathcal{C}$ puede escalar a espacios de estados y acción grandes con aproximación de políticas mediante funciones, π_{θ} .

Aprendizaje por Imitación Generativo Adversario

El algoritmo del Aprendizaje por Imitación Generativo Adversario (GAIL) se basa en ir aplicando la siguiente fórmula del gradiente de políticas para el objetivo de aprendizaje por refuerzo inverso para una política parametrizada.

$$\nabla_{\theta} \max_{c \in \mathcal{C}} \mathbb{E}_{\pi_{\theta}}[c(s, a)] - \mathbb{E}_{\pi_E}[c(s, a)] = \nabla_{\theta} \max_{c \in \mathcal{C}} \mathbb{E}_{\pi_{\theta}}[c^*(s, a)] = \mathbb{E}_{\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(a|s) \mathcal{Q}_{c^*}(s, a)] \quad (3.10)$$

dónde

- $c^* = \arg \max_{c \in \mathcal{C}} \mathbb{E}_{\pi_{\theta}}[c(s, a)] - \mathbb{E}_{\pi_E}[c(s, a)]$

$$\blacksquare Q_{c^*}(\bar{s}, \bar{a}) = \mathbb{E}_{\pi_\theta}[c^*(\bar{s}, \bar{a}) | s_0 = \bar{s}, a_0 = \bar{a}]$$

Obsérvese que se está considerando que la función de costo óptima es c^* y se utiliza el gradiente para la actualización de la política parametrizada π_θ . Por tanto, utilizamos el gradiente para actualizar la política parametrizada π_θ en base a la función de costo óptima c^* .

Este enfoque permite aprender una política que imite el comportamiento de un experto al buscar maximizar la recompensa esperada de acuerdo con la función de costo óptima.

¿Dónde está la conexión con las GANs?

Anteriormente, se han estudiado dos tipos de regularizadores y aquí se propone un regularizador de costo que combina tanto aprendizaje por imitación en entornos grandes como una conexión directa con las redes generativas adversarias.

Se ha visto cómo los regularizadores constantes nos conducen a algoritmos de aprendizaje por imitación donde las medidas de ocupación coinciden, pero son inmanejables en entornos grandes. Por otro lado, se ha estudiado que los regularizadores de las funciones indicadoras para el conjunto de funciones de costo lineales, denotado como $\mathcal{C}_{linear}$, lleva a algoritmos donde las medidas de ocupación no coinciden exactamente, pero sí son manejables en entornos grandes.

Se propone el siguiente regularizador que combina lo mejor de las propuestas anteriores: el regularizador constante y el regularizador de las funciones indicadoras. Asimismo, tiene la ventaja de que permite cualquier función de costo siempre que sea negativa.

$$\psi_{GA} \triangleq \begin{cases} \mathbb{E}_{\pi_E}[g(c(s, a))] & \text{si } c \leq 0 \\ +\infty & \text{en otro caso} \end{cases} \quad (3.11)$$

donde

$$g(x) = \begin{cases} -x - \log(1 - \exp x) & \text{si } x \leq 0 \\ +\infty & \text{en otro caso} \end{cases} \quad (3.12)$$

La elección de ψ_{GA} está motivada por el resultado que se obtiene cuando calculamos la conjugada convexa, ψ_{GA}^* .

$$\psi_{GA}^*(\rho_\pi - \rho_{\pi_E}) = \max_{D \in (0,1)^{\mathcal{S}x\mathcal{A}}} \mathbb{E}_\pi[\log(D(s, a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s, a))] \quad (3.13)$$

donde el discriminador es un clasificador $D : \mathcal{S}x\mathcal{A} \rightarrow (0, 1)$.

La ecuación que acabamos de visualizar es la pérdida logarítmica negativa óptima del problema de clasificación binaria para distinguir entre los pares estado-acción de π y π_E .

La expresión de esta pérdida coincide con la divergencia de Jensen-Shannon que se define $D_{JS}(\rho_\pi, \rho_{\pi_E}) \triangleq D_{KL}(\rho_\pi || (\rho_\pi + \rho_{\pi_E})/2) + D_{KL}(\rho_{\pi_E} || (\rho_\pi + \rho_{\pi_E})/2)$ y es una métrica entre distribuciones.

Considerando la entropía causal H como una regularización de políticas controlada por el valor $\lambda \geq 0$, obtenemos un nuevo algoritmo de aprendizaje por imitación que trata de encontrar la política cuya medida de ocupación minimiza la divergencia de Jensen-Shannon entre su medida de ocupación y la del experto. Seguidamente, vemos la expresión de como se llega a este algoritmo, conocido como **Aprendizaje por Imitación Generativo Adversario (GAIL)**.

$$\begin{aligned} & \min_{\pi} \psi_{GA}^*(\rho_{\pi} - \rho_{\pi_E}) - \lambda H(\pi) & (\text{GAIL}) \\ & = \min_{\pi} \left(\max_{D \in (0,1)^{\mathcal{S} \times \mathcal{A}}} \mathbb{E}_{\pi}[\log(D(s, a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s, a))] \right) - \lambda H(\pi) \\ & = D_{JS}(\rho_{\pi}, \rho_{\pi_E}) - \lambda H(\pi) \end{aligned}$$

En nuestro entorno, la medida de ocupación del aprendizaje ρ_{π} es análoga a la distribución generada por G , y la medida de ocupación del experto ρ_{π_E} es análoga a la distribución de datos reales.

3.2.5. Algoritmo GAIL

El siguiente algoritmo resuelve el problema planteado en la ecuación anterior. Su objetivo es encontrar un punto de silla $(\pi, \mathcal{D})$ en la expresión:

$$\mathbb{E}_{\pi}[\log(D(s, a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s, a))] - \lambda H(\pi) \quad (3.14)$$

y resolver el siguiente problema min-max

$$\min_{\pi} \max_{D \in (0,1)^{\mathcal{S} \times \mathcal{A}}} \mathbb{E}_{\pi}[\log(D(s, a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s, a))] - \lambda H(\pi) \quad (3.15)$$

Para hacerlo, primero introducimos la aproximación de funciones para π y $\mathcal{D}$: ajustaremos la política parametrizada π_{θ} , con pesos θ , y una red Discriminadora $D_{\omega} : \mathcal{S} \times \mathcal{A} \rightarrow (0, 1)$, con pesos ω .

Algorithm 13: Algoritmo GAIL

Para cada trayectoria i

1. Actualizamos los parámetros del Discriminador de ω_i a ω_{i+1} con el gradiente:

$$\mathbb{E}_{\pi_i}[\nabla_{\omega} \log(D_{\omega}(s, a))] + \mathbb{E}_{\pi_E}[\nabla_{\omega} \log(1 - D_{\omega}(s, a))] \quad (3.16)$$

2. Tomamos la política π_{θ_i} y actualizamos la política $\pi_{\theta_{i+1}}$ utilizando TRPO con función de coste $\log(D_{\omega_{i+1}}(s, a))$. Realizamos la actualización del gradiente:

$$\mathbb{E}_{\pi_i}[\nabla_{\theta} \log \pi_{\theta}(a|s) \mathcal{Q}(s, a)] - \lambda \nabla_{\theta} H(\pi_{\theta}) \quad (3.17)$$

donde $\mathcal{Q}(\bar{s}, \bar{a}) = \mathbb{E}_{\theta_i}[\log(D_{\omega_{i+1}}(s, a)) | s_0 = \bar{s}, a_0 = \bar{a}]$

4. Hibridación Q-Learning (HQL)

En este capítulo, se presenta una nueva propuesta que trata de aplicar una GANs sobre una base de datos de tablas Q obtenidas previamente para ampliar la información disponible durante el proceso de aprendizaje con el entorno en el algoritmo Q-Learning. Definimos un modelo generador diferente al modelo generador de GAIL ya que le permitimos que tenga acceso al entorno. De esta forma, esta técnica no pertenece al aprendizaje por imitación sino mas bien a una hibridación entre el paradigma del aprendizaje por imitación y el aprendizaje por refuerzo.

4.1. Fundamentos

Hibridación Q-Learning (HQL) es una técnica que conecta las GANs con la hibridación establecida entre el aprendizaje por imitación y el algoritmo Q-Learning del aprendizaje por refuerzo. Demuestra cómo se puede entrenar una GAN para que su generador produzca tablas Q que se usen durante el entrenamiento con el entorno.

Para implementar HQL, en primer lugar ejecutamos n episodios del algoritmo Q-Learning para construir un conjunto de datos de n tablas Q que formará nuestra base de datos. La primera tabla se denota como $Q(S, A, 1)$, la segunda tabla se denota como $Q(S, A, 2), \dots$, la n -ésima tabla se denota como $Q(S, A, n)$.

Así, se notará a nuestra **base de datos del modelo GAN**, conocida también como “base de datos experta o real” donde tenemos todas las tablas

$$Q(S, A) = \{Q(S, A, 1), Q(S, A, 2), \dots Q(S, A, n)\}, \forall (S, A) \in \mathcal{S} \times \mathcal{A}$$

n es el número de episodios, $Q(S, A, i)$ es la matriz obtenida en el episodio i que tiene tantas filas como tipo de observaciones discretas tengamos en $\mathcal{S}$ y tantas columnas como tipo de acciones discretas tengamos en $\mathcal{A}$.

Nuestro agente utilizará la base de datos experta en su entorno para poder extraer de ella, observaciones y acciones expertas, esto es, para cada observación del entorno, S , el agente decide realizar la acción, A , que maximiza su recompensa a partir de su tabla $Q(S, A, i)$ en el episodio i .

De esta forma, nuestra GAN entrenada obtendrá un generador capaz de producir distintas tablas sintéticas $Q^*(S, A, i)$ y formar $Q^*(S, A) = \{Q^*(S, A, 1), Q^*(S, A, 2), \dots Q^*(S, A, n)\}$ con tablas muy similares a las pertenecientes a la base de datos experta. Entonces generará matrices con las cuales un agente decidirá la acción que maximiza su recompensa a partir de una observación de su entorno. Si hemos dicho que estas matrices, las que conforman la base de datos experta y las matrices producidas con el generador, tienen

cierto grado de similitud entonces podemos pensar que las acciones que escogerá el agente cuando se encuentre la misma observación del entorno también tendrán cierta similitud.

Por tanto, este planteamiento de generación sintética de tablas Q-Learning, también nos estaría generando una política que se aproxima a la política experta. Dicho de otro modo, la generación sintética de la función Q-Learning a través del modelo generativo de las GANs estaría adquiriendo conocimiento a partir de la política experta para construir la tabla Q^* a partir de la política que imita a la política experta. Claramente, tenemos un trasfondo de un algoritmo de aprendizaje por imitación aunque como se ha comentado exista cierta hibridación con el aprendizaje por refuerzo.

4.2. Una perspectiva distribucional del aprendizaje por refuerzo

Antes que nada, tenemos que recordar que estamos trabajando con espacios discretos así que las definiciones propuestas son sobre variables aleatorias discretas. Vamos a enmarcar nuestro planteamiento a través de conceptos necesarios de variables aleatorias y vectores aleatorios.

Observación 4. La σ -álgebra de $\mathcal{S} \times \mathcal{A} \times \mathbb{N}$, se define como $\mathcal{S} \times \mathcal{A} \times \mathbb{N}$ ya que es la menor σ -álgebra que contiene a todos los conjuntos de la forma $S \times A \times n$ con $S \in \mathcal{S}$, $A \in \mathcal{A}$ y $n \in \mathbb{N}$, y es cerrada bajo las operaciones de unión, intersección y complemento.

$$\sigma(\{S \times A \times n : S \in \mathcal{S}, A \in \mathcal{A}, n \in \mathbb{N}\}) := \mathcal{S} \times \mathcal{A} \times \mathbb{N}$$

Definición 4.1. Una **variable aleatoria** Q sobre un espacio probabilístico base $(\mathcal{S} \times \mathcal{A} \times \mathbb{N}, \mathcal{S} \times \mathcal{A} \times \mathbb{N}, P)$ es una función medible unidimensional, definida sobre un espacio probabilístico, esto es, $Q : (\mathcal{S} \times \mathcal{A} \times \mathbb{N}, \mathcal{S} \times \mathcal{A} \times \mathbb{N}, P) \rightarrow (\mathbb{R}, \mathcal{B})$ tal que

$$Q^{-1}(B) \in \mathcal{S} \times \mathcal{A} \times \mathbb{N}, \forall B \in \mathcal{B}$$

donde $Q^{-1}(B) = \{(s, a, i) \in \mathcal{S} \times \mathcal{A} \times \mathbb{N} : Q(s, a, i) \in B\}$, equivalentemente,

$$Q^{-1}((-\infty, x]) \in \mathcal{S} \times \mathcal{A} \times \mathbb{N}, \forall x \in \mathbb{R}$$

Definición 4.2. Si Q es una variable aleatoria sobre $(\mathcal{S} \times \mathcal{A} \times \mathbb{N}, \mathcal{S} \times \mathcal{A} \times \mathbb{N}, P)$, se define su **distribución de probabilidad** como una función $P_Q : \mathcal{B} \rightarrow [0, 1]$, definida por

$$P_Q(B) := P_Q(\{(s, a, i) \in \mathcal{S} \times \mathcal{A} \times \mathbb{N} : Q(s, a, i) \in B\}), \forall B \in \mathcal{B}$$

Definición 4.3. La **función de distribución de una variable aleatoria unidimensional** es una función $F_Q : \mathbb{R} \rightarrow [0, 1]$, definida por

$$F_Q(x) = P_Q((-\infty, x]) = P(Q \in (-\infty, x]), \forall x \in \mathbb{R}$$

y satisface las siguientes propiedades:

- Monótona no decreciente

- *Continua a la derecha*
- $\exists \lim F_Q(x), \lim_{x \rightarrow -\infty} F_Q(x) = 0$ y $\lim_{x \rightarrow \infty} F_Q(x) = 0$

A partir de la definición de variable aleatoria, vamos a definir un vector aleatorio.

Se ha adoptado la siguiente **notación de variable aleatoria** $Q_i(s, a) := Q(s, a, i)$ para poder definir las componentes de un vector aleatorio de la forma mas clara posible. Esto es, el subíndice i se corresponde con el número de episodio. Q_i es una variable aleatoria sobre un espacio probabilístico $(\mathcal{S} \times \mathcal{A}, \mathcal{S} \times \mathcal{A}, P)$.

Definición 4.4. *Un **vector aleatorio**, también conocido como variable aleatoria multidimensional, es una función medible multidimensional, definida sobre un espacio probabilístico; esto es, una función:*

$$Q := (Q_1, Q_2, \dots, Q_n)^T : (\mathcal{S} \times \mathcal{A}, \mathcal{S} \times \mathcal{A}, P) \rightarrow (\mathbb{R}^n, \mathcal{B}^n) \text{ tal que,}$$

$$Q^{-1}(B) \in \mathcal{S} \times \mathcal{A}, \forall B \in \mathcal{B}^n$$

donde $Q^{-1}(B) = \{(s, a) \in \mathcal{S} \times \mathcal{A} : Q(s, a) \in B\}$, equivalente,

$$\begin{aligned} Q^{-1}((-\infty, x_1] \times (-\infty, x_2] \times \dots \times (-\infty, x_n]) = \\ \{(s, a) \in \mathcal{S} \times \mathcal{A} / Q_1(s, a) \leq x_1, Q_2(s, a) \leq x_2, \dots, Q_n(s, a) \leq x_n\} \end{aligned}$$

Podemos definir el teorema de medibilidad donde se relacionan los vectores aleatorios con las variables aleatorias

Teorema 4.1. *Sea $(\mathcal{S} \times \mathcal{A}, \mathcal{S} \times \mathcal{A}, P)$ un espacio probabilístico y $Q := (Q_1, Q_2, \dots, Q_n)^T : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^n$ una función n -dimensional, entonces*

$$Q \text{ es un vector aleatorio} \iff Q_1, Q_2, \dots, Q_n \text{ son variables aleatorias}$$

Definición 4.5. *Si Q es un vector aleatorio sobre un espacio probabilístico $(\mathcal{S} \times \mathcal{A}, \mathcal{S} \times \mathcal{A}, P)$, se define su **distribución de probabilidad** como una función $P_Q : \mathcal{B}^n \rightarrow [0, 1]$, definida por*

$$P_Q(B) = P((s, a) \in \mathcal{S} \times \mathcal{A} : Q(s, a) \in B), \forall B \in \mathcal{B}^n$$

Nótese que podemos abreviar $\{(s, a) \in \mathcal{S} \times \mathcal{A} : Q(s, a) \in B\}$ como $\{Q \in B\}$ y definir $P_Q(B) = P(Q \in B)$

4.3. Formulación

Nuestra base de datos de la GAN definida como $Q(S, A)$, base de datos real, contiene muestras de la distribución de probabilidad P_Q , $Q(S, A, n) \sim P_Q$, $n \in \mathbb{N}$, y como son nuestra GAN tiene como objetivo generar muestras sintéticas $Q^*(S, A, n)$, de una distribución de probabilidad P_{Q^*} , $Q^*(S, A, n) \sim P_{Q^*}$, $n \in \mathbb{N}$, que sean similares a las muestras reales, debemos de cuantificar la similitud a través de una función de distancia. Para medir la distancia entre distribuciones de probabilidad de variables aleatorias discretas en una GAN, podemos estudiar distintas medidas. Un medida muy común que vimos en el estudio del enfoque básico de las GANs es la divergencia de Jensen-Shannon (JSD).

Definición 4.6. El enfoque básico de las GANs para generar sintéticamente tablas Q-Learning consta del siguiente generador y discriminador.

- El **generador** consiste en una función multidimensional:

$$G := Q^* = (Q_1^*, \dots, Q_n^*)^T : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^n, \forall n \in \mathbb{N}.$$

Claramente, cumple las propiedades de vector aleatorio así que trabajaremos con su distribución de probabilidad, P_{Q^*} , que es desconocida a priori, queremos determinar su distribución, P_{Q^*} , de forma que sea una buena aproximación de P_Q .

- Para conseguirlo, introducimos un **discriminador**, esto es, una función que a cada tabla Q-Learning le asigne la probabilidad de que sea real o sintética,

$$D : Q^*(S, A) \cup Q(S, A) \rightarrow [0, 1]$$

donde $(S, A) \subset \mathcal{S} \times \mathcal{A}$.

El enfoque básico sabemos que se conoce como Vanilla GAN y que se puede formular mediante un problema min-max entre un generador y un discriminador. A continuación, presentamos la formulación que trata de resolver este problema específico:

$$\begin{aligned} \min_G \max_D V(D, G) &= \min_{Q^*} \max_D V(D, Q^*) \\ &= \min_{Q^*} \max_D (\mathbb{E}_{Q(S, A) \sim P_Q} [\log D(Q(S, A))] + \mathbb{E}_{Q^*(S, A) \sim P_{Q^*}} [\log(1 - D(Q^*(S, A)))] \end{aligned} \quad (4.1)$$

4.4. Algoritmo de entrenamiento HQL

Para resolver el problema de optimización presentado con la ecuación 4.1 necesitamos aplicar el descenso del gradiente estocástico (SGD). El discriminador puede aproximarse como una red neuronal, $D := D_\omega$ donde ω son los pesos que aprende la red discriminadora y el generador al ser un vector aleatorio, solo se calcula su función de pérdida con objeto de que vaya disminuyendo a lo largo del entrenamiento.

Por tanto, tenemos el siguiente algoritmo muy similar al algoritmo presentado en el capítulo de las GANs con la diferencia de que aquí tenemos un generador encargado de producir tablas Q-Learning sintéticas. Sea m el número de episodios que se ha definido para ejecutar el algoritmo Q-Learning, entonces tenemos el siguiente pseudocódigo:

Algorithm 14: Algoritmo HQL

1. **Para** un número de iteraciones de entrenamiento **hacer**

a) **Para** k pasos **hacer**

- 1) Tomar una muestra de m ejemplos, $Q^* = \{Q_1^*, \dots, Q_m^*\}$ de la distribución P_{Q^*} , base de datos sintética construida por el generador.
- 2) Tomar una muestra de m ejemplos, $Q = \{Q_1, \dots, Q_m\}$ del conjunto de entrenamiento que sigue una distribución P_Q , base de datos experta.
- 3) Actualizar el discriminador D_ω por ascenso de su gradiente estocástico con respecto a ω :

$$\nabla_\omega \frac{1}{m} \sum_{i=1}^m [\log D_\omega(Q) + \log(1 - D_\omega(Q^*))]$$

- 4) Actualizar la función de pérdida del discriminador

$$\mathcal{L}_D = \frac{1}{m} \sum_{i=1}^m [\log D_\omega(Q) + \log(1 - D_\omega(Q^*))]$$

b) **fin**

- c) Tomar una muestra de m ejemplos, $Q^* = \{Q_1^*, \dots, Q_m^*\}$ de la distribución P_{Q^*} , base de datos sintética construida por el generador.

- d) Actualizar la función de pérdida del generador

$$\mathcal{L}_G = \frac{1}{m} \sum_{i=1}^m [\log(1 - D_\omega(Q^*))]$$

2. **fin**

Nos habremos dado cuenta que solo tenemos una red neuronal, entonces solo aplicamos el algoritmo del gradiente descendente estocástico sobre el discriminador. Esto tiene la ventaja de que permite un procesamiento mucho más rápido y finalmente, conseguiremos un discriminador que no sepa distinguir las tablas Q^* sintéticas de las Q reales.

Cuando se consiga esto y bajo este marco descrito, habremos encontrado un generador definido como una variable aleatoria, $G = Q^*$, donde su distribución de probabilidad es la mas similar a la distribución de probabilidad de la base de datos experta. Por ende, será capaz de generar tablas Q^* de la que podremos extraer una política similar a la política experta de nuestra base de datos real Q .

Parte II.

Experimentación y resultados

5. Métodos y herramientas

En este capítulo se va a explicar la metodología que se va a seguir para la parte práctica de este trabajo. Se describirá qué algoritmos se han escogido y sobre qué entornos se van a ejecutar. Asimismo, se detallarán las implementaciones que se han diseñado para GAIL y HQL, así como la explicación de un software conocido como Sinergym con el que pondremos en práctica algunas metodologías e implementaciones.

5.1. Metodología

Con el objeto de evaluar la efectividad de las técnicas descritas, GAIL e HQL, seleccionamos diferentes entornos donde podamos probar su aplicabilidad.

Inicialmente, comenzaremos en el Capítulo 6 con entornos proporcionados por Gym OpenAI, ya que incluye tareas clásicas de control, lo que nos permitirá poner a prueba nuestras implementaciones. Decidimos aplicar el algoritmo GAIL en los entornos CartPole y Taxi de Gym. Sin embargo, el algoritmo HQL solo pudimos probarlo con el entorno Taxi debido a que requiere que los espacios de observaciones y acciones sean discretos, y el entorno CartPole presenta observaciones continuas. Dado que el entorno Taxi se utilizó en ambas técnicas, realizamos una comparativa entre ellas en el Capítulo 8.

Una vez que confirmamos que nuestras implementaciones de GAIL e HQL funcionaban correctamente, decidimos adaptarlas a un entorno más complejo: Sinergym. Este entorno presenta diversas simulaciones de edificios diseñadas para controlar sus sistemas HVAC mediante agentes de aprendizaje por refuerzo. Las observaciones en este caso son variables relacionadas con el sistema HVAC, y las acciones corresponden a modificaciones del sistema. Para las pruebas, hemos seleccionado tres edificios denominados 5Zone, DataCenter y Warehouse, los cuales se describen detalladamente en el Capítulo 7 donde abordamos la experimentación con Sinergym. Es importante señalar que en este entorno las observaciones son continuas, por lo que solo pudimos probar el algoritmo GAIL.

La Tabla 5.1 contiene un resumen de los entornos con los que hemos trabajado y con qué técnicas se han probado. Así, podemos conocer el tipo de observaciones y de acciones que tiene cada entorno, ya sean continuas o discretas.

Entornos	Observaciones		Acciones		Algoritmo	
	Discreto	Continuo	Discreto	Continuo	GAIL	HQL
Taxi - Gym	×		×		×	×
CartPole - Gym		×	×		×	
5Zone - Sinergym		×	×		×	
Datacenter - Sinergym		×	×		×	
Warehouse - Sinergym		×	×		×	

Tabla 5.1.: Experimentación

GAIL y HQL siguen el siguiente proceso de entrenamiento y evaluación de modelos. Primero, estas técnicas generan los datos de la base de datos con la que vamos a entrenar a nuestro modelo generativo, seguidamente, se sumergen en el proceso de entrenamiento donde cada n épocas el modelo generador es validado y por último, se valida la actuación del discriminador.

1. Generar datos de entrenamiento para construir la base de datos experta
2. Entrenar a nuestro modelo a lo largo de M épocas
 - a) Cada n épocas

$\{ \text{EVALUAR AL GENERADOR (AGENTE) EN EL ENTORNO (X10)} \}$
3. Evaluar al Discriminador haciendo uso de la entropía cruzada binaria

Cuando evaluamos al generador, obtenemos 10 interacciones entre el Agente y el entorno, así que decidimos ver la gráfica de estas 10 interacciones. La idea de visualizar todas las gráficas es apreciar la existencia de un comportamiento robusto.

Por otro lado, cada vez que evaluamos al generador, calculamos la media de estas 10 recompensas de cada gráfica y decidimos mostrar una gráfica de como van variando las recompensas medias obtenidas cada vez que evaluamos el comportamiento del generador, esto es, el comportamiento del agente. Así, con esta gráfica si podemos percibir como de bueno ha sido el aprendizaje ya que con las anteriores solo percibíamos un comportamiento robusto entre ellas. Lo ideal es que conforme aumente las épocas de entrenamiento, el valor de recompensas medias vaya aumentando así el agente cada vez que se evalúe tomará decisiones cuyas recompensas medias van aumentando.

5.2. Implementación

Vamos a ver como implementar las técnicas GAIL e HQL para posteriormente aplicarlas a entornos de Gym OpenAI y a entornos de Sinergym.

5.2.1. Implementación GAIL

A continuación, describimos las implementaciones realizadas para el discriminador, generador y cómo integrarlas de manera específica en un modelo GAN con objeto de construir nuestro GAIL. La Figura 5.1 ilustra la construcción GAIL.

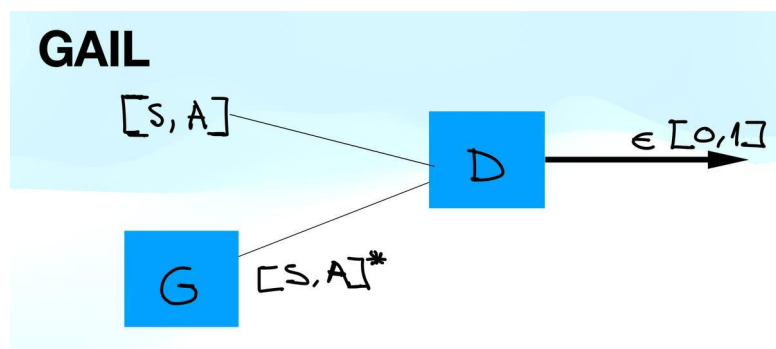


Figura 5.1.: Esquema GAIL: aplicamos una GAN a una base de datos de secuencias de estados-acciones expertas

■ Discriminador

Tenemos una clase llamada Discriminator que contiene como atributos:

- La red neuronal **discriminator_net** que se encarga de asignar una probabilidad entre 0 y 1 a la entrada verdadera $[s, a]$ de la base de datos experta o a la entrada falsa $[s, a]^*$. Básicamente, su arquitectura consiste en un perceptrón multicapa con una capa de entrada que recibe secuencias de estados-acciones de la forma (None, NUMERO_ACCIONES+NUMERO_OBSERVACIONES). Después de la capa de entrada, tenemos cuatro capas totalmente conectadas que utilizan la función de activación ReLU y una capa de salida con una neurona que tiene a la función sigmoide como función de activación.
- La secuencia de estados-acciones expertas $[s, a] = [[s_0, a_0], [s_1, a_1], [s_2, a_2], \dots, [s_n, a_n]]$ se almacenan en la variable **expert_s_a**.
- La secuencia de estados-acciones sintéticas, obtenidas con el generador, $[s, a]^* = [[s_0, a_0]^*, [s_1, a_1]^*, [s_2, a_2]^*, \dots, [s_m, a_m]^*]$ se almacenan en la variable **agent_s_a**.
- La probabilidad que le asigna la red neuronal discriminator_net a la secuencia $[s, a]$ experta es **prob_expert**. Esto es, la probabilidad de que la secuencia sea la secuencia experta, se obtiene calculando la salida de la red neuronal del

Discriminador cuando recibe como entrada la secuencia de estados-acciones expertas.

- La probabilidad que le asigna la red neuronal `discriminator_net` a la secuencia $[s, a]^*$ sintética se encuentra en el atributo **prob_agent**. Este valor se obtiene cuando introducimos en la red neuronal del Discriminador, la secuencia de estados-acciones sintética como entrada de la red.
- El atributo **rewards** define la recompensa obtenida cuando el agente realiza $[s, a]^*$.

Los métodos que contiene son simples métodos `get`.

La función de pérdida del Discriminador que va a usar nuestra GAIL se implementa en la función **loss_fn_D** que a partir de las probabilidades `prob_agent` y `prob_expert` es capaz de calcular el valor de la función de pérdida del modelo Discriminador. Esta función de pérdida se utilizará en el entrenamiento de GAIL ya que actualizaremos los parámetros de la red neuronal en función del gradiente de esta función de pérdida.

Básicamente, esta función implementa la fórmula de la entropía cruzada binaria que es una función de pérdida muy comúnmente utilizada en problemas donde el objetivo es asignar una muestra a una de las dos clases posibles.

Primero, calcula la esperanza del logaritmo de la probabilidad de la clase real y la esperanza del logaritmo de la probabilidad de la clase falsa. Luego, suma ambas pérdidas y las niega (-loss) para obtener la pérdida final a minimizar.

- **Generador** La clase `Policy_net` implementa la clase del generador donde tenemos los siguientes atributos:

- El entorno **env** sobre el que va a actuar el agente.
- El algoritmo PPO, conocido como algoritmo de Optimización de Política Proximal, combina ideas del algoritmo A2C, que utiliza múltiples trabajadores, y del algoritmo TRPO, donde se utiliza una región de confianza para mejorar al actor.

Este algoritmo se almacena en el atributo **model** y cada generador `Policy_net` tendrá un algoritmo PPO encargado de actualizar políticas. PPO se declara para nuestro entorno `env` y se entrena con un total de 25 pasos.

- La observación **obs** se utiliza como observación de entrada a una red neuronal, *generator_next_Act*, que producirá su acción consecuente. Esta acción será de la que partirá nuestro entorno para producir el estado inicial s_0 . Nótese que la red neuronal solo se utiliza con esta observación y que esta observación no es s_0 , mas bien es un estado anterior a s_0 .
- La probabilidad asignada a cada acción que podría ejecutar nuestro agente cuando se encuentra en el estado `obs` es **act_probs**. Esta probabilidad se calcula haciendo uso de la red neuronal `generator_net_Act`, cuya arquitectura es un perceptrón multicapa con cuatro capas totalmente conectadas que reciben

como entrada un tensor de la forma (None,NUMERO OBSERVACIONES). Todas las capas tienen la función de activación tanh a excepción de la última capa que produce dos unidades de salida con la función de activación softmax, una unidad por cada tipo de acción.

- La evaluación de la política que persigue Policy_net se almacena en **v_preds**, es la función de valor de la política Policy_net y le asignamos la salida de una red neuronal *generator_net_v_preds* que recibe como entrada la observación obs.

Esta red neuronal también tiene la arquitectura de un perceptrón multicapa, formado a partir de 3 capas totalmente conectadas, recibiendo entradas de la forma (None,NUMERO_OBSERVACIONES) y produciendo como salida un único valor, la aproximación de la función valor de la política.

- La acción determinística **act_determinist** es la acción de act_probs que tiene asignada una probabilidad superior.
- La acción estocástica se almacena en **act_stochastic** y se define a partir del logaritmo de act_probs.

En cuanto a los métodos, tenemos métodos get, un método act que en función de si el argumento que se le pase a la función, devuelve la acción estocástica o determinística y por último, el método generate_fakes que merece una explicación mas detallada ya que es el encargado de generar las secuencias $[s, a]^* = [[s_0, a_0]^*, [s_1, a_1]^*, [s_2, a_2]^*, \dots, [s_m, a_m]^*]$

El método **generate_fakes** realiza la ejecución de un número de episodios donde en cada episodio comprueba si la recompensa total es superior a un umbral. Cada experimento tendrá un número concreto de episodios almacenado en una variable global EPISODES. En el episodio donde se supere el umbral, nos quedamos con su conjunto de estados-acciones para formar la secuencia de $[s, a]^*$ sintéticas con la que intentaríamos engañar al Discriminador. Claramente, si se termina el entorno, se finaliza la generación de la secuencia falsa.

La ejecución de un episodio concreto en este método consiste en ir prediciendo acciones a partir del estado en el que se encuentra el agente haciendo uso de nuestro modelo PPO. Por tanto, si nuestro agente se encuentra en un estado s_i , ejecutará la acción $a_i = model.get_predict(s_i)$ y tendremos la secuencia $[s_i, a_i]^*$ que formará parte del $[s, a]^*$ del episodio.

```

1 Policy = Policy_net('policy',env, obs=[next_obs])
2 # Por cada steps en cada episodio
3 while terminated!= True and truncated!= True:
4
5     run_policy_steps += 1
6
7     #s_i-1=next_obs
8     action, states_oc = Policy.get_model().predict(next_obs) #a_i
9

```

```

10     next_obs, reward, terminated, truncated, info = env.step(action) #
      s_i, R_i
11

```

Este método devuelve las observaciones, las acciones, la política antigua y la política actual. Considerando la política actual como la política que ha generado la secuencia $[s, a]^*$ y la política antigua como la política que generó la secuencia en el episodio anterior.

A continuación, veremos como se establece una conexión entre Generator y Policy_net en GAIL. También, declararemos una instancia de PPOTrain en el entrenamiento de GAIL para ir entrenando las diferentes Policy_net con la misión de conseguir un PPO que prediga las acciones según cada estado en el que se encuentre el agente.

La función de pérdida de Policy_net (Ecuación 1.11) se implementa en **loss_fn_ppo** y describe la función de pérdida del modelo PPO (sección 1.4.5) almacenado en nuestra clase.

```

1 def loss_fn_ppo(act_probs, act_probs_old, gaes, clip_value=0.2):
2     #--> Construir loss_clip
3
4     ratios = tf.exp(tf.math.log(tf.clip_by_value(act_probs, 1e-10,
5         1.0)) - tf.math.log(tf.clip_by_value(act_probs_old, 1e-10, 1.0)))
6
7     clipped_ratios = tf.clip_by_value(ratios, clip_value_min=1 -
8         clip_value, clip_value_max=1 + clip_value)
9     loss_clip = tf.minimum(tf.multiply(gaes, ratios), tf.multiply(
10         gaes, clipped_ratios))
11     loss_clip = tf.reduce_mean(loss_clip)
12
13     # minimizar -loss == maximizar loss
14     loss = -loss
15
16     return loss

```

Algoritmo de Aprendizaje por Refuerzo: PPO

Como se ha comentado en la sección de GAIL del Capítulo 3, nuestro generador tendrá un TRPO que trata de aprender e imitar a la política experta de forma que cuando finalicemos el entrenamiento de nuestra GAIL, tendremos un generador con un TRPO capaz de generar secuencias $[s, a]^*$ parecidas a $[s, a]$ y por tanto, se obtendrá una política similar a la política experta. Dicho de otro modo, si nuestro agente ejecuta las secuencias $[s, a]^*$ sobre nuestro entorno, obtendrá una imitación del comportamiento que realizaría nuestro agente con las secuencias $[s, a]$. Esto es, GAIL aproximaría una política similar a la política experta que sigue nuestro agente en el entorno.

En la literatura se ha estudiado la implementación de GAIL con TRPO pero en esta experimentación se propone el uso de PPO ya que ha demostrado ser eficaz en una amplia

gama de tareas de aprendizaje por refuerzo y ha logrado resultados muy significativos en entornos complejos y desafiantes.

El algoritmo PPO (Proximal Policy Optimization), descrito en la sección 1.4.5, es un algoritmo de aprendizaje por refuerzo utilizado para entrenar agentes de inteligencia artificial en entornos de toma de decisiones. Fue propuesto por OpenAI en 2017 y su objetivo es entrenar a un agente para que tome decisiones óptimas en un entorno, maximizando la recompensa acumulada a lo largo del tiempo. El agente va aprendiendo a través de ensayo y error, actualizando su política y mejorando su comportamiento paso a paso.

Este algoritmo se basa en la optimización de políticas mediante el método del descenso del gradiente ascendente.

Asimismo, combina ideas del algoritmo A2C, que utiliza múltiples trabajadores, y del algoritmo TRPO, donde se utiliza una región de confianza para mejorar al actor. Por tanto, podemos usarlo perfectamente en lugar de usar TRPO.

Cada generador `Policy_net` tiene un modelo PPO, entrenado a lo largo de 25 steps.

La clase **PPOTrain** almacena en sus atributos las políticas `Policy` y `Old_Policy` que se corresponden con las políticas $\pi_{\theta_{i+1}}$ y π_{θ_i} . También almacena como atributo sus pesos pero actualizando los pesos ya que una vez declaremos `Policy`, esta pasará a ser la política i (`Old_Policy`) y la nueva política que se declare será la $i + 1$ (`Policy`). Por tanto, le asignamos $\theta_i = \theta_{i+1}$ ya que ajustaremos unos nuevos pesos θ_{i+1} . Esta clase presenta otros atributos necesarios para el cálculo de la función de pérdida del generador, como es el valor `gaes` (generative advantage estimator).

En cuanto a los métodos, tenemos un método donde se llama a la función de pérdida de PPO y este método se denota **loss_fn.G** para así indicar que devuelve el valor de la función de pérdida del generador.

Modelo generativo y de aprendizaje por imitación: GAIL

Como se ha comentado en la sección de teoría, la técnica GAIL es una modificación de una GAN donde se alteran las funciones de pérdida, el modelo generador y discriminador para que lleve adelante nuestro objetivo de generar una política que imite a la política experta.

Procedemos a explicar cómo se ha construido nuestra GAIL que utilizará las funciones de pérdidas explicadas anteriormente junto con Discriminator como modelo discriminador y `Policy_net` como generador. Como métrica se utiliza la media de todos los valores, como optimizador del Discriminador se aplicará Adam con un valor de 0.001 de tasa de aprendizaje (`learning_rate`) mientras que el Generador no utiliza ningún tipo de optimizador ya que en GAIL definimos un objeto de la clase `PPOTrain` con la política actual y la antigua; esta instancia será la encargada de actualizar los pesos de la política actual sin necesidad de construir un optimizador. Si la política antigua π_{θ_i} tiene unos pesos θ_i y los pesos de la política actual $\pi_{\theta_{i+1}}$ son θ_{i+1} , entonces $\theta_{i+1} \rightarrow \theta_i$ y la política actual pasa a ser la antigua ya que en el siguiente paso del entrenamiento se construirá una nueva política cuando generemos la secuencia $[s, a]^*$

■ Constructor

```

1 def __init__(self, discriminator, generator):
2     super().__init__()
3     self.discriminator = discriminator
4     self.generator=Policy_net
5     self.d_loss_metric = keras.metrics.Mean(name="d_loss")
6     self.g_loss_metric = keras.metrics.Mean(name="g_loss")

```

■ Compilación

Como debe preservar la estructura de una GAN y una GAN solo tiene una función de pérdida que comparten ambos modelos, aquí se le indica la función de pérdida del discriminador pero en el entrenamiento del generador se toma la función de pérdida de PPO.

```

1 def compile(self,d_optimizer , loss_fn_D ):
2     super(GAN, self).compile(run_eagerly=True)
3     self.d_optimizer = d_optimizer
4     self.loss_fn_D= loss_fn_D

```

■ Entrenamiento

A partir del generador se crea la secuencia de $[s, a]^*$ sintéticas

```

1 generate_observations , generate_actions , rewards , Old_Policy ,
   Policy=self.generator.generate_fakes(self.generator)
2 generate_a_one_hot=np.eye(env.action_space.n)[generate_actions
   ]
3
4 if generate_observations.shape[0] == generate_a_one_hot.shape
   [0]:
5     dataset_gen = np.concatenate([generate_observations ,
   generate_a_one_hot], axis=1)
6 else:
7     return
8     print('N de acciones generadas!=N estados generados')
9     generate_a_one_hot_resized = np.resize(generate_a_one_hot ,
   generate_observations.shape)
10    dataset_gen = np.concatenate([generate_observations ,
   generate_a_one_hot_resized], axis=1)

```

Cuando estén creadas se combinarán con las secuencias expertas para poder entrenar al Discriminador con las secuencias expertas etiquetadas como 1 y las secuencias sintéticas etiquetadas como 0.

```

1 len_real = X_train.shape[0]
2 len_fakes=dataset_gen.shape[0]
3 # Las etiquetas de las imagenes combinadas las tenemos que
   crear nosotros introduciendo algo de ruido con tf.random.
   uniform

```

```

4 labels = tf.concat(
5     [tf.ones((len_real, 1)), tf.zeros((len_fakes, 1))],
6     axis=0
7 )
8
9 labels += 0.05 * tf.random.uniform(tf.shape(labels))

```

1. Entrenamiento del Discriminador

Nuestro modelo discriminador predice etiquetas para el dataset combinado y calculamos el valor de la función de pérdida del Discriminador, `d_loss`, a partir de esas predicciones y de las etiquetas verdaderas definidas en `labels`.

```

1 with tf.GradientTape() as tape:
2     predictions=np.zeros((len_real+len_fakes,num_obs+
3         num_action))
4     predictions = self.discriminator.discriminator_net(
5         combined_images)
6
7     d_loss = self.loss_fn_D(labels, predictions)

```

Con este valor podemos calcular el gradiente de la función de pérdida y aplicar el optimizador del Discriminador, Adam, para que en cada paso del entrenamiento se vaya disminuyendo el gradiente y lleguemos a un mínimo.

```

1 grads = tape.gradient(d_loss, self.discriminator.getNet().
2     trainable_weights)
3
4 self.d_optimizer.apply_gradients(
5     zip(grads, self.discriminator.getNet().trainable_weights)
6 )

```

2. Entrenamiento del Generador

Inicialmente, debemos de declarar una instancia de `PPOTrain` con las políticas obtenidas al generar la secuencia $[s, a]^*$, esto es, con la política actual y la antigua. Después accediendo al método `loss_fn_G` de la instancia de `PPOTrain` tenemos el valor de la función de pérdida para nuestro generador.

```

1 # Se realiza todo en la clase PPOTrain
2 ppotrain=PPOTrain(Policy,Old_Policy,actions=
3     generate_actions,rewards=rewards, obs=
4     generate_observations[0])
5
6 with tf.GradientTape() as tape:
7     g_loss = ppotrain.loss_fn_G()

```

■ Evaluación del Discriminador

Consiste en generar secuencias sintéticas $[s, a]^*$ y evaluar nuestra red neuronal de nuestro discriminador tanto con esas secuencias sintéticas como con los datos

reales almacenados en `XTest`. Previamente, deberemos compilar la red neuronal indicándole la función de pérdida que se desea optimizar y el optimizador Adam.

```

1 self.discriminator.discriminator_net.compile(optimizer=self.
    d_optimizer, loss=self.loss_fn_D, metrics=['accuracy'])
2
3 # Evaluamos como CNN
4 loss_real, acc_real=self.discriminator.discriminator_net.
    evaluate(X_test[0:len_real], tf.ones((len_real,1)),
    verbose=1)
5
6 loss_fake, acc_fake=self.discriminator.discriminator_net.
    evaluate(dataset_gen[0:len_fakes], tf.ones((len_fakes,1)),
    verbose=1)

```

Con esta evaluación podremos comparar el valor de la función de pérdida que se obtiene cuando nuestro discriminador trata de evaluar secuencias generadas y secuencias reales. Estos valores nos servirán para determinar cómo de bueno ha sido nuestro entrenamiento.

■ Evaluación del Generador

Consiste en probar si las secuencias sintéticas $[s, a]^*$ que genera nuestro generador `Policy_net` es capaz de seguirlas un agente en nuestro entorno. Para ello, realizamos una ejecución de 10 episodios, esto es, restauramos nuestro entorno 10 veces y ponemos a nuestro agente a que interactúe a través de las acciones predichas por el modelo PPO almacenado en nuestro generador. Cuando finaliza el episodio, mostramos la recompensa total acumulada.

Particularidades de la configuración GAIL en Sinergym

■ Discriminador

A continuación, mostramos la arquitectura de la red neuronal **discriminator_net** cuya misión es asignar una probabilidad en el intervalo $[0, 1]$ a cada entrada $[s_i, a_i]$.

- **5ZONE**: Se crea un perceptrón multicapa utilizando la biblioteca keras cuyo nombre es `discriminator_net`. La construcción del modelo parte de una capa de entrada de tamaño (None, 27) donde 27 es la suma del tamaño de observaciones y del tamaño de acciones. Este perceptrón multicapa se construye a partir de 3 capas ocultas de 14 unidades cada una, utilizando la función de activación ReLU. Su capa de salida tiene una única neurona con una función de activación sigmoide para que produzca la probabilidad que la entrada sea real o falsa.
- **DATACENTER**: Este entorno presenta un número de tipo de observaciones superior respecto al 5Zone así que se propone una arquitectura un poco más sofisticada donde incluyamos 6 capas ocultas en un perceptrón multicapa. Su capa de entrada tiene un tamaño de (None, 39) ya que recibe las secuencias de estados-acciones falsas y verdaderas. Estas seis capas ocultas utilizan la

función de activación ReLU y la salida de la red, se obtiene con una capa densa con una función de activación sigmoide al igual que hicimos en 5Zone.

- **WAREHOUSE**

En este edificio volvemos a diseñar un perceptrón multicapa con 3 capas ocultas como hicimos con 5Zone, que utiliza una función de activación sigmoide en su capa de salida. Este perceptrón recibe capas de entrada del tamaño (None, 33) y sus capas ocultas tienen una función de activación RELU.

- **Variables globales**

Como variables globales tenemos el número de épocas, $EPOCHS = 100$, y el tamaño de batch, $BATCH_SIZE = 28032$. Tenemos una variable llamada $EPISODES$ y que toma el valor de 1, marca el número de episodios que entrenará a nuestro modelo de aprendizaje por refuerzo para generar secuencias de estados-acciones falsas.

La generación de secuencias de observaciones-acciones sintéticas sería mucho más potente si pudiésemos definir la variable $EPISODES$ superior a 1 pero se necesitaría más recursos computacionales, se queda como línea futura a desarrollar.

5.2.2. Implementación HQL

Se procede a estudiar la implementación de la técnica Hibridación Q-Learning.

La intuición de esta técnica surge de la definición de las tablas Q , que representan la recompensa que de media obtiene un agente cuando sigue una determinada política, π_i , para una acción y estado de entrada. Por tanto, una tabla Q^* que obtenga recompensas similares para otra política, π_j , bajo la misma acción y estado, tendrá que aproximarse a π_i .

Bajo esta idea, podemos usar la configuración original de la GAN como podemos ver en el siguiente esquema:

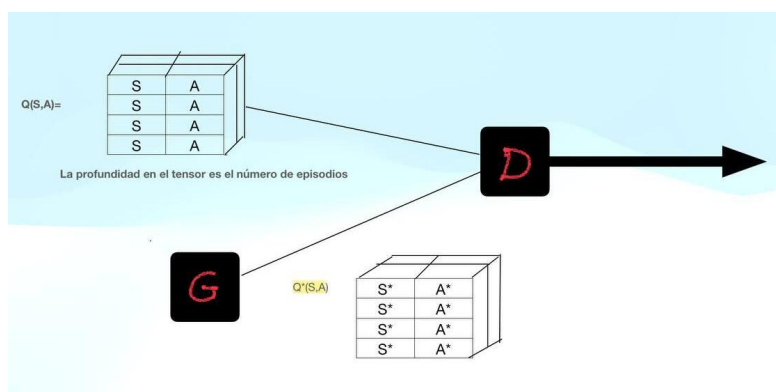


Figura 5.2.: Esquema HQL: aplicamos una GAN a una base de datos experta de tablas Q-Learning

A continuación, se describen las implementaciones realizadas para el discriminador y el generador, y cómo integrarlas en el modelo GAN para conseguir nuestro objetivo: construir un generador capaz de generar tensores de la forma $Q^*(S, A) = \{Q^*(S, A, 1), \dots, Q^*(S, A, M)\}$ siendo M el número de episodios que ejecutará el agente en la construcción de las tablas Q^* sintéticas.

Por consiguiente, nuestro modelo generará M tablas sintéticas, a partir de las cuales podremos extraer M políticas individuales para cada estado s . Cuando presentemos un estado s a nuestro agente, ejecutaremos la acción $a = \arg \max Q^*(s, \cdot)$.

■ Discriminador

Definimos una red neuronal convolucional (CNN) llamada **discriminator** que recibe entradas $Q(S, A)$ de la forma (NUMERO_OBSERVACIONES, NUMERO_ACCIONES, 1) y produce la probabilidad de que la tabla $Q(S, A)$ sea verdadera. Esta red está formada por dos capas convolucionales con función de activación LeakyReLU y finalmente introducimos una capa totalmente conectada junto con Dropout. A continuación, veamos su arquitectura.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 250, 3, 64)	640
leaky_re_lu (LeakyReLU)	(None, 250, 3, 64)	0
conv2d_1 (Conv2D)	(None, 125, 2, 128)	73856
leaky_re_lu_1 (LeakyReLU)	(None, 125, 2, 128)	0
flatten (Flatten)	(None, 32000)	0
dropout (Dropout)	(None, 32000)	0
dense (Dense)	(None, 1)	32001
=====		
Total params: 106,497		
Trainable params: 106,497		
Non-trainable params: 0		

Figura 5.3.: HQL: Red neuronal convolucional del discriminador

■ Generador

Definimos una clase para el generador llamada **Q-learning** y recibe este nombre porque nuestro generador construirá tablas Q mediante el algoritmo Q-learning, notadas como Q^* .

Sus atributos son el entorno, el número de episodios, la tabla Q inicializada a cero, la recompensa media, la recompensa total y un vector de recompensas donde almacenaremos la recompensa obtenida en la ejecución de Q-learning para cada episodio. El número de episodios nos indicará cuantas tablas Q sintéticas generaremos con el Generador ya que en cada episodio el algoritmo Q-Learning obtiene una tabla Q de estados y acciones. La tabla Q se inicializa a cero pero conforme el Generador vaya creando tablas Q^* sintéticas, almacenará esos valores.

En cuanto a los métodos, tenemos métodos get y el método `Q_create` encargado de construir $Q^*(S, A) = \{Q^*(S, A, 1), \dots, Q^*(S, A, M)\}$ donde M es el número de episodios que recibe como argumentos. Vamos a explicar paso a paso su implementación para entender correctamente su construcción.

1. Para cada episodio se obtiene una aproximación $Q(\text{estados}, \text{acciones})$ con el algoritmo Q-learning
 - a) Seleccionar la mejor acción para el estado actual, considerando los valores de la variable tabla Q .
 - b) Ejecutar la acción a través del entorno y pasar al estado siguiente, obtener la recompensa.
 - c) Actualización de Q con el algoritmo Q-learning y ajustando $\alpha = 0.7$.
 - d) Actualización de la recompensa acumulada en ese episodio y del estado siguiente. Esto es, sumarle a la recompensa acumulada la recompensa obtenida en este paso de tiempo (step).
2. Cuando finaliza el episodio i , introducimos la tabla Q en datos y cuando se ejecuten todos los episodios devolvemos esos datos. Así, se construye $Q^*(S, A, i)$ para introducirse en el tensor de tablas Q sintéticas $Q^*(S, A)$.

■ Funciones de pérdida

Se va a utilizar la entropía cruzada binaria para el discriminador y nos quedamos con la parte de las Q^* generadas de la entropía cruzada binaria para definir la pérdida del generador ya que son con las que trabaja nuestro modelo. Por tanto, podemos definir las siguientes funciones de pérdida:

```

1 def discriminator_loss(real_img, fake_img):
2     real_loss = tf.reduce_mean(real_img)
3     fake_loss = tf.reduce_mean(fake_img)
4     return fake_loss - real_loss
5
6 def generator_loss(fake_img):
7     return -tf.reduce_mean(fake_img)
```

Como optimizador se utilizará Adam con una tasa de aprendizaje (learning rate) de 0.001, β_1 de 0.5 y β_2 de 0.9.

Modelo generativo: GANs

Utilizaremos el esquema original de la GAN donde trabajaremos con el generador y discriminador que hemos definido anteriormente.

- Constructor

El constructor **init** se encarga de inicializar el discriminador y el generador.

- Compilación

El método **compile** trata de compilar el modelo GAN y este modelo presenta dos funciones de pérdida diferentes, una para el discriminador y otra para el generador. Como optimizador se usará la configuración de Adam que se ha definido anteriormente.

- Entrenamiento

1. Entrenamiento del Discriminador

- La función `tf.GradientTape()` la denotamos como `tape` y nos servirá para calcular el gradiente. Esta función es útil para implementar el algoritmo de retropropagación en el entrenamiento de redes neuronales.

```
1 with tf.GradientTape() as tape:
2
```

- Vamos a generar la tabla Q^* falsa y obtener tanto los logits de las tablas Q real como de las Q^* sintéticas que acabamos de generar. Esto se consigue evaluando al discriminador con las dos tablas

```
1
2 fakes=self.generator.Q_create_t(epochs=batch_size)
3 fakes = tf.stack(fakes)
4 fake_logits=self.discriminator(fakes)
5 real_logits = self.discriminator(X_train)
6
```

- A partir de los logits calculamos la pérdida del discriminador.

```
1 d_cost = self.d_loss_fn(real_img=real_logits, fake_img
2                       =fake_logits)
```

- Finalmente, se calcula el gradiente de la función de pérdida y actualizamos los pesos del discriminador usando su optimizador.


```

1 grads = tape.gradient(d_loss, self.discriminator.
    trainable_weights)
2
3 self.d_optimizer.apply_gradients(
4     zip(grads, self.discriminator.trainable_weights)
5 )
6

```

2. Entrenamiento del Generador

- Generamos Q^* falsa y obtenemos sus logits.

```

1 with tf.GradientTape() as tape:
2     # Generamos la Q* falsa a traves del generador
3     fakes = self.generator.Q_create_t()
4     fakes = tf.stack(fakes)
5     # Obtenemos los logits del discriminador en la Q*
        falsa
6     gen_img_logits = self.discriminator(fakes)

```

- Calculamos la pérdida del generador a partir de los logits

```

1 g_loss = self.g_loss_fn(gen_img_logits)

```

- No calculamos gradiente de la función de pérdida del generador ni actualizamos sus pesos ya que solo nos interesa en esta propuesta obtener un generador capaz de producir tablas Q^* similares a las tablas Q . El papel del generador es básicamente aplicar el algoritmo Q-learning para generar la entrada falsa al modelo Discriminador haciendo uso de aprendizaje por refuerzo.

■ Evaluación del Discriminador

Se genera una tabla Q^* a partir de los datos de evaluación, después se compila la red neuronal y se evalúa para las Q^* sintéticas y para las Q reales

■ Evaluación del Generador

Consiste en ver como actúe el agente siguiendo la política obtenida con las tabla generadas Q^* de evaluación. Esto es, para cada s' nuestro agente realiza la acción a donde se encuentre el máximo $\max Q(s', a)$. ($a = \arg \max Q(s', .)$)

5.3. Sinergym

Sinergym permite la creación de entornos con los que poder interactuar entre un agente externo y un motor de simulación de edificios, en este caso *EnergyPlus*. De esta forma, se pueden controlar diferentes componentes del edificio durante dicha simulación, como el HVAC. El control externo puede ser realizado por cualquier cosa que respete la interfaz de Gymnasium, como es el caso de los algoritmos de aprendizaje por refuerzo profundo.

Aquí vamos a explicar muy brevemente la herramienta Sinergym. Puede documentarse mejor en los siguientes enlaces [Sinb] y [Sina].

Sinergym incluye distintos edificios con los que podemos trabajar y aplicar el estudio teórico realizado en este trabajo. En nuestra experimentación se ha optado por utilizar los siguientes edificios para abordar el desarrollo práctico.

- **5Zone:** consiste en un edificio de una sola planta dividido en 1 zona interior y 4 zonas exteriores. Su superficie es de $463.6m^2$ y se encuentra equipado con un paquete VAV (esto es, bobina de enfriamiento DX y bobinas de calefacción de gas) con entrada de tamaño automático completo que serán de controladas por el sistema HVAC. Figura 5.4
- **DataCenter:** Es un edificio de $491.3m^2$ que presenta dos zonas asimétricas. Cada zona cuenta con un sistema HVAC que consta de economizadores de aire, enfriadores evaporativos, bobinas de enfriamiento DX, bobinas de agua helada y VAV. La principal fuente de calor proviene de los servidores alojados. Figura 5.5
- **Warehouse:** Es un edificio de planta no residencial de $4598 m^2$, dividido en tres zonas: almacenamiento a granel, almacenamiento fino y una oficina. La zona de oficinas está cerrada en dos lados y en la parte superior por la zona de almacenamiento fino, y es la única zona con ventanas. Los tipos de combustible disponibles son gas y electricidad, y está equipado con un sistema de HVAC (calefacción, ventilación y aire acondicionado). Figura 5.6

El edificio **5Zone** se decide tomar del entorno *Eplus-5Zone-hot-discrete-v1* que presenta un espacio de 17 variables de observaciones que toman valores reales continuos y 10 tipos de acciones discretas compuestas por dos variables cada una, pertenecientes a los valores de temperatura de frío y calor. Las observaciones las podemos visualizar en la Tabla 5.3 y las acciones en la Tabla 5.2. Las acciones son cambios de los valores de calefacción-ventilación, mas bien, sería ubicar la temperatura de HVAC en los valores que le corresponda según la acción que se seleccione del 1 al 10. Por ejemplo, si seleccionamos la acción A_1 , entonces nuestro agente ubicará el valor de temperatura de frío del sistema HVAC a 15 grados y la temperatura de calor a 30 grados.

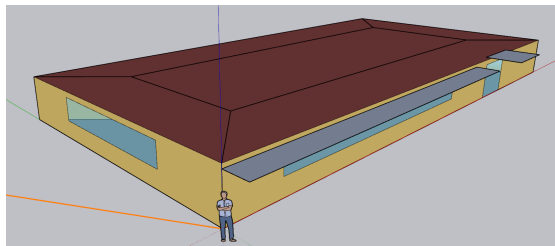


Figura 5.4.: Edificio 5Zone [Sina]

Acción	Valores
A_1	(15, 30)
A_2	(16, 29)
A_3	(17, 28)
A_4	(18, 27)
A_5	(19, 26)
A_6	(20, 25)
A_7	(21, 24)
A_8	(22, 23)
A_9	(22, 22)
A_{10}	(21, 21)

Tabla 5.2.: Acciones 5Zone

S_1 : Temperatura exterior del aire (Entorno)
S_2 : Humedad relativa exterior del aire (Entorno)
S_3 : Velocidad del viento (Entorno)
S_4 : Dirección del viento (Entorno)
S_5 : Tasa de radiación solar difusa por área (Entorno)
S_6 : Tasa de radiación solar directa por área (Entorno)
S_7 : Temperatura de consigna de calefacción del termostato de la zona (ESPACIO1-1)
S_8 : Temperatura de consigna de refrigeración del termostato de la zona (ESPACIO1-1)
S_9 : Temperatura del aire de la zona (ESPACIO1-1)
S_{10} : Humedad relativa del aire de la zona (ESPACIO1-1)
S_{11} : Cantidad de ocupantes en la zona (ESPACIO1-1)
S_{12} : Temperatura del aire de los ocupantes (ESPACIO1-1 PERSONAS1)
S_{13} : Demanda total de electricidad HVAC de la instalación (Edificio completo)
S_{14} : Año
S_{15} : Mes
S_{16} : Día
S_{17} : Hora

Tabla 5.3.: Observaciones 5Zone

En segundo lugar, se estudia el edificio **Datacenter** y trabajaremos con el entorno *Eplus-datacenter-hot-discrete-v1* ya que nos presenta una estructura de estados y acciones similar a la anterior. Esto se debe a que contiene un espacio de observaciones reales continuo y un espacio de acciones discretos. A continuación, se van a mostrar las 29 tipos de observaciones continuas en la Tabla 5.4. En este caso las 10 acciones discretas se corresponden con las acciones anteriores del control del HVAC en el edificio 5Zone, Tabla 5.2.

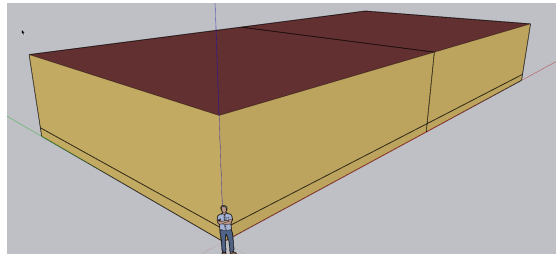


Figura 5.5.: Edificio Datacenter [Sina]

Estado	Tipo de estado
S_1	Temperatura exterior del aire (Entorno)
S_2	Humedad relativa exterior del aire (Entorno)
S_3	Velocidad del viento (Entorno)
S_4	Dirección del viento (Entorno)
S_5	Tasa de radiación solar difusa por área (Entorno)
S_6	Tasa de radiación solar directa por área (Entorno)
S_7	Temperatura de consigna de calefacción del termostato de la zona (Zona Oeste)
S_8	Temperatura de consigna de refrigeración del termostato de la zona (Zona Oeste)
S_9	Temperatura del aire de la zona (Zona Oeste)
S_{10}	Temperatura media de radiación térmica del confort térmico de la zona (Zona Oeste PERSONAS)
S_{11}	Humedad relativa del aire de la zona (Zona Oeste)
S_{12}	Valor de vestimenta del confort térmico de la zona (Zona Oeste PERSONAS)
S_{13}	Modelo Fanger de PPD del confort térmico de la zona (Zona Oeste PERSONAS)
S_{13}	Cantidad de ocupantes en la zona (Zona Oeste)
S_{14}	Temperatura del aire de los ocupantes (Zona Oeste PERSONAS)
S_{15}	Temperatura de consigna de calefacción del termostato de la zona (Zona Este)
S_{16}	Temperatura de consigna de refrigeración del termostato de la zona (Zona Este)
S_{17}	Temperatura del aire de la zona (Zona Este)
S_{18}	Temperatura media de radiación térmica del confort térmico de la zona (Zona Este PERSONAS)
S_{19}	Humedad relativa del aire de la zona (Zona Este)
S_{20}	Valor de vestimenta del confort térmico de la zona (Zona Este PERSONAS)
S_{21}	Modelo Fanger de PPD del confort térmico de la zona (Zona Este PERSONAS)
S_{22}	Recuento de ocupantes de personas de la zona (Zona Este)
S_{23}	Temperatura del aire de los ocupantes (Zona Este PERSONAS)
S_{24}	Demanda total de electricidad HVAC de la instalación (Edificio completo)
S_{25}	Año
S_{26}	Mes
S_{27}	Día
S_{28}	Hora

Tabla 5.4.: Observaciones Datacenter

Como tercera línea de estudio, tenemos el edificio **Warehouse** donde podemos visualizar en las siguientes tablas como se estructuran las 23 tipos de observaciones continuas (Tabla 5.6), y las 10 acciones discretas (Tabla 5.5). Anteriormente las acciones solo consistían en ubicar los valores de calefacción-ventilación de la zona, ahora como tenemos tres zonas debemos de ajustar los valores de calefacción-ventilación de la oficina y del almacenamiento a granel, sin embargo del almacenamiento a granel solo se necesita

del valores de calefacción. Las acciones ahora tomarán valor en 5 campos en lugar de en 2 campos: los valores de calefacción-ventilación de la oficina, los valores de calefacción-ventilación del almacenamiento fino y el valor de calefacción de almacenamiento a granel.

Anteriormente, los edificios aplicaban un control del sistema HVAC uniforme y este edificio tiene un control independizado por cada zona.

Acción	Valores
A_1	(15, 30, 15, 30, 15)
A_2	(16, 29, 16, 29, 16)
A_3	(17, 28, 17, 28, 17)
A_4	(18, 27, 18, 27, 18)
A_5	(19, 26, 19, 26, 19)
A_6	(20, 25, 20, 25, 20)
A_7	(21, 24, 21, 24, 21)
A_8	(22, 23, 22, 23, 22)
A_9	(22, 22, 22, 22, 23)
A_{10}	(21, 21, 21, 21, 24)

Tabla 5.5.: Acciones Warehouse

S_1 : Temperatura seca del aire exterior (Entorno)
S_2 : Humedad relativa del aire exterior (Entorno)
S_3 : Velocidad del viento en el exterior (Entorno)
S_4 : Dirección del viento en el exterior (Entorno)
S_5 : Tasa de radiación solar difusa por área en el exterior (Entorno)
S_6 : Tasa de radiación solar directa por área en el exterior (Entorno)
S_7 : Temperatura de consigna de calefacción del termostato (Zona 1 - Oficina)
S_8 : Temperatura de consigna de refrigeración del termostato (Zona 1 - Oficina)
S_9 : Temperatura del aire en la zona (Zona 1 - Oficina)
S_{10} : Humedad relativa del aire en la zona (Zona 1 - Oficina)
S_{11} : Recuento de ocupantes en la zona (Zona 1 - Oficina)
S_{12} : Temperatura de consigna de calefacción del termostato (Zona 2 - Almacenamiento Fino)
S_{13} : Temperatura de consigna de refrigeración del termostato (Zona 2 - Almacenamiento Fino)
S_{14} : Temperatura del aire en la zona (Zona 2 - Almacenamiento Fino)
S_{15} : Humedad relativa del aire en la zona (Zona 2 - Almacenamiento Fino)
S_{16} : Temperatura de consigna de calefacción del termostato (Zona 3 - Almacenamiento a Granel)
S_{17} : Temperatura del aire en la zona (Zona 3 - Almacenamiento a Granel)
S_{18} : Humedad relativa del aire en la zona (Zona 3 - Almacenamiento a Granel)
S_{19} : Tasa total de demanda de electricidad del HVAC del edificio (Edificio completo)
S_{20} : Año
S_{21} : Mes
S_{22} : Día
S_{23} : Hora

Tabla 5.6.: Observaciones Warehouse

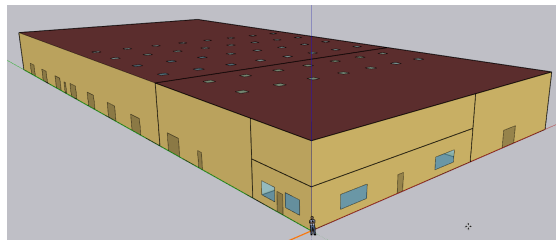


Figura 5.6.: Edificio Warehouse [Sina]

6. Experimentación con entornos de OpenAI GYM

En este capítulo, describiremos los experimentos realizados sobre entornos de OpenAI Gym [TTK⁺23], de este modo trabajaremos con una librería de Python desarrollada por OpenAI que permite implementar diferentes algoritmos de Aprendizaje por Refuerzo y la simulación entre el Agente y el Entorno. Básicamente, los entornos son el bloque fundamental de OpenAI Gym e implementan en Python un simulador del entorno en el cual queremos entrenar nuestro Agente. Nuestro foco de estudio van a ser dos entornos de OpenAI Gym: CartPole y Taxi. Los códigos implementados para el desarrollo de la experimentación de este capítulo se pueden encontrar en <https://github.com/silviabm98/GymOpenAI>

6.1. Entorno de ejecución

A continuación, vamos a ir nombrando las herramientas que se han utilizado para realizar los experimentos con CartPole y Taxi.

- Se ha usado Google Colab y el lenguaje de programación Python para llevar a cabo toda la experimentación. Google Colab, también conocido como Colaboratory, es un producto de Google Research que permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador. Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación. La decisión de usar Google Colab ha sido por las ventajas que presenta ya que nos proporciona una GPU gratuita, permite almacenar cuadernos Python en google Drive, no necesita hacer ninguna configuración y permite compartir el código de forma fácil.
- En cuanto a bibliotecas de Python, se ha usado Keras para el diseño de las redes neuronales usadas en los distintos experimentos. Keras es una biblioteca de redes neuronales de código abierto en Python que vamos a ejecutar sobre Tensorflow y está diseñada para hacer posible la experimentación con redes de Aprendizaje Profundo. Tensorflow es una biblioteca de código libre para la computación numérica, desarrollada por Google Brain para aplicaciones de aprendizaje automático y de redes neuronales profundas, y que permite principalmente procesar fácilmente matrices o tensores. Esta biblioteca provee herramientas de alto nivel donde nos encontramos con Keras. Podemos observar que si se quiere programar una nueva red neuronal o realizar cálculos muy costosos con matrices se deberá usar Ten-

sorflow pero para usar alguna red neuronal de las que ya existen es mejor usar Keras.

- Para importar nuestros entornos hemos tenido que importar la librería de Python Gym y Gymnasium desarrollada por OpenAI. Gym permite implementar diferentes algoritmos de aprendizaje por refuerzo y simular la interacción entre agentes y entornos.
- Para poder utilizar PPO necesitamos importar la biblioteca stable_baselines3 la cual ofrece implementaciones de algoritmos de aprendizaje por refuerzo en Pytorch. Además, proporciona una interfaz fácil de usar y eficiente para entrenar y evaluar agentes de aprendizaje por refuerzo en una variedad de entornos.

6.2. Descripción

Inicialmente, empezamos con el entorno Cartpole, realizando un experimento con aprendizaje por imitación generativo adversario (GAIL) cuya intención es aprender una política que imite a la política experta. La política experta viene definida en forma de secuencias estado-acción, $[s, a] \in \mathcal{S} \times \mathcal{A}$, $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$, para el espacio de estados $\mathcal{S}$ y espacio de acciones $\mathcal{A}$ de CartPole.

El entorno Taxi se va a probar con el aprendizaje por imitación generativo adversario (GAIL) y con Hibridación Q-Learning (HQL) para poder comparar ambas técnicas.

La experimentación con GAIL se realiza de forma similar a CartPole con la salvedad de que en este entorno tenemos observaciones discretas pero no le vamos a aplicar ningún tipo de preprocesamiento especial. Así que se utilizará el código explicado para CartPole pero cambiando el entorno para aplicar la técnica GAIL.

Con respecto a HQL, se va a experimentar con una GAN con el objetivo de que el generador aprenda a generar tablas Q^* que se parezcan a la tabla Q obtenida con la política experta. Recordemos que Q es una tabla o función de acción-valor para una política, entonces lo que queremos es obtener la función de acción-valor asociada a la política que más se parezca a la función acción-valor de la política experta.

La base de datos que utiliza el modelo GAN para aprender esta formada por tensores 3D con el siguiente tamaño (n^o de estados, n^o de acciones, n^o de episodios) $= (500, 6, N)$. Mas bien, tenemos tantas tablas $Q(S, A, i)$ como episodios ejecutemos (recordemos la notación de la base de datos experta en la sección 4.1, donde i indica la tabla obtenida en el episodio i del algoritmo Q-Learning y se utiliza para enumerar a la tabla de dicho episodio), esto es nuestro tensor de entrada será $Q(S, A) = \{Q(S, A, 1), \dots, Q(S, A, N)\}$ siendo N número de episodios. La profundidad del tensor viene definida por el número de episodios que realizará nuestro agente en este entorno.

6.3. CartPole

Este entorno corresponde con la versión del problema CartPole descrito por Barto, Sutton, and Anderson en [BSA83]. Sintetizando, este entorno consiste en un poste que

está unido mediante una articulación no accionada a un carro que se mueve a lo largo de una pista sin fricción. El péndulo se coloca verticalmente sobre el carro y el objetivo es equilibrar el poste aplicando fuerzas en dirección izquierda y derecha sobre el carro. Nuestro espacio de *acciones* es discreto de tamaño dos y tenemos espacio de *estados* continuo de tamaño cuatro. Para cada acción 0 o 1, donde 0 significa empujar el carro hacia la izquierda y 1 empujar el carro hacia la derecha, tenemos un estado asociado a dicha acción. Este estado almacena 4 valores correspondiente a la posición del poste, velocidad del poste, ángulo del poste y velocidad angular del poste.

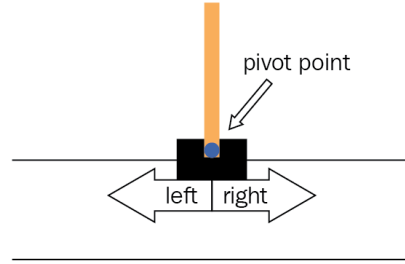


Figura 6.1.: Entorno clásico de control *CartPole* [TTK⁺23]

6.3.1. Configuración

La política experta viene definida en dos archivos csv: *observations.csv* y *actions.csv*. El archivo *observations.csv* tiene entradas de cuatro valores donde la entrada i se corresponde con $s_i = \{s_{i1}, s_{i2}, s_{i3}, s_{i4}\}$ y en *actions.csv* cada entrada tiene un único valor correspondiente con a_i . La secuencia de acciones se preprocesa mediante una codificación uno-en-caliente (OneHotEncoder) y obteniendo que $a_i = \{a_{i1}, a_{i2}\}$ con $a_{i1}, a_{i2} \in \{0, 1\}$, $i \in \{1, 2, \dots, n\}$ donde n es el número de secuencias que tenemos. Concatenando la información de *observations.csv* con el preprocesamiento de las acciones, tenemos las secuencias $[s_i, a_i] = \{s_{i1}, s_{i2}, s_{i3}, s_{i4}, a_{i1}, a_{i2}\} \forall i$ de la política experta. Así, tenemos la construcción de nuestro dataset con el que entrenaremos nuestro modelo GAIL.

La política experta es la política que selecciona la acción para cada observación del entorno utilizando un modelo de aprendizaje por refuerzo preentrenado. En este caso, se ha utilizado un PPO preentrenado con un total de 25000 pasos y para cada s_i predice a_i , así construimos la secuencia experta de estados-acciones $[s_i, a_i]$.

Consideramos una división del 80 % del dataset para Training y 20 % para Test, entonces entrenaremos con 3200 instancias y realizaremos la evaluación con 800 instancias.

6.3.2. Ejecución

Se ha realizado un entrenamiento a lo largo de 100 épocas con un optimizador Adam que utiliza una tasa de aprendizaje de 0.001. Cada 10 épocas de entrenamiento se evaluaba al generador para ver la capacidad que tenía nuestro agente de producir secuencias

de estados-acciones e interactuar con el entorno a base de esas secuencias sintéticas a lo largo de 10 episodios, calculándose la recompensa media de esos 10 valores.

6.3.3. Resultados

Examinamos Tabla 6.1 con el objetivo de extraer conclusiones basadas en los datos obtenidos en este experimento.

Se aprecia como lentamente la pérdida del Discriminador va disminuyendo conforme va aumentando el número de épocas de entrenamiento mientras que la pérdida del Generador tiene pequeñas oscilaciones pero acaba aumentando su valor y acercándose a cero ya que comienza con un valor negativo. Finalmente, comparando los valores de la primera época y de la última podemos afirmar que el generador ha aumentado un poco su pérdida y que el discriminador ha disminuido notablemente su pérdida, aproximándose ambos más a cero.

Observando el comportamiento de la política que se está aprendiendo con GAIL, podemos afirmar que la política va mejorando conforme aumentan las épocas de entrenamiento. No hay un crecimiento lineal, pero si comparamos la recompensa media de la última época de entrenamiento, la época 100, con la época 10, se obtiene un aumento de 8.5 en la recompensa del agente. Por tanto, la política que se ha aprendido en este proceso de entrenamiento imita mejor a la política experta que la política PPO inicial y se ha obtenido un buen aprendizaje por imitación.

Epochs	Loss D	Loss G	Mean Reward
1	0.7966	-1.2702	-
10	0.7895	-0.9318	20.1
20	0.7905	-0.9836	28.7
30	0.7783	0.9918	30.5
40	0.8247	-0.5667	25.2
50	0.7860	-0.8441	32.3
60	0.7341	-1.0046	19.7
70	0.7492	-1.3616	31.2
80	0.7515	-1.0207	31.2
90	0.7058	-0.9918	21.4
100	0.6966	-0.9902	28.6

Tabla 6.1.: Cartpole: Entrenamiento GAIL

Además, el discriminador ha aprendido a lo largo del entrenamiento y podemos evaluar este aprendizaje. En la Tabla 6.2 vemos los resultados obtenidos de la función de pérdida cuando evaluamos al discriminador con datos nuevos, XTest, entonces obtenemos la pérdida “Loss real” y sobre datos generados, entonces obtenemos la pérdida “Loss fakes”.

Acabamos de apreciar como nuestro modelo ha aprendido a distinguir algunas secuencias reales $[s, a]$ de las secuencias sintéticas $[s, a]^*$.

Loss real	Loss fakes
0.69539	0.75012

Tabla 6.2.: Cartpole: Evaluación Discriminador GAIL

Anteriormente, se ha comentado que en el proceso del entrenamiento de nuestras técnicas se ha realizado una evaluación del generador cada 10 épocas así que se procede a ilustrar las distintas gráficas ordenadas de la época 10 a la 100, de izquierda a derecha y de arriba a abajo.

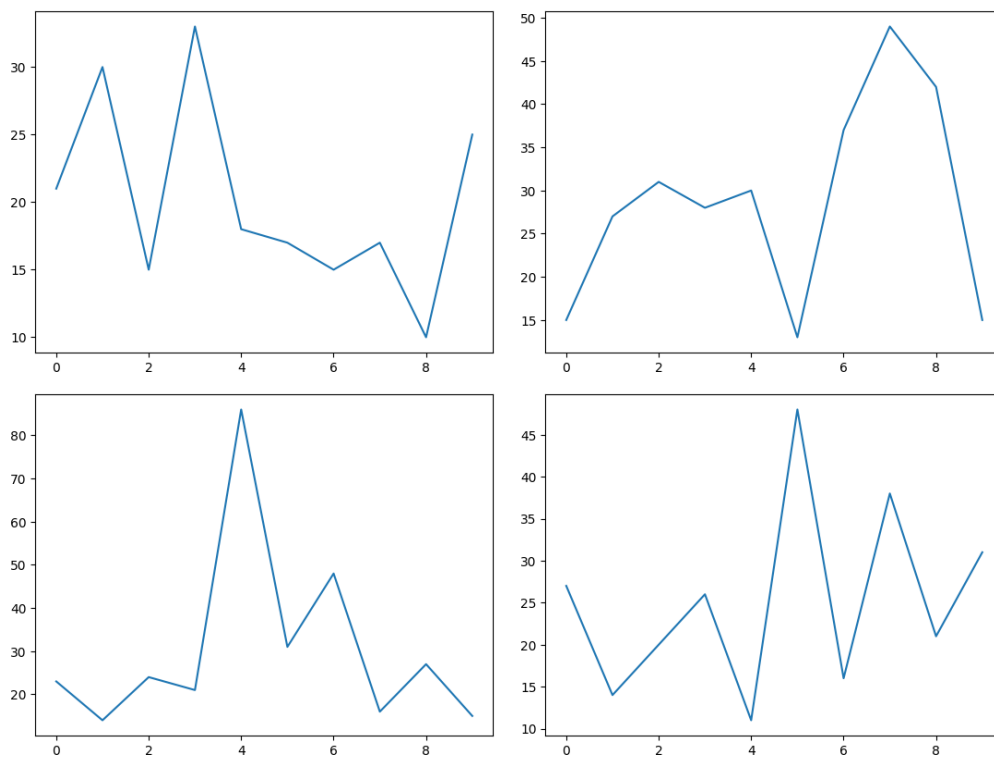


Figura 6.2.: Cartpole: Evaluación del Generador GAIL en las épocas 10, 20, 30, 40 (Eje x-Nº episodio, Eje y-Recompensa)

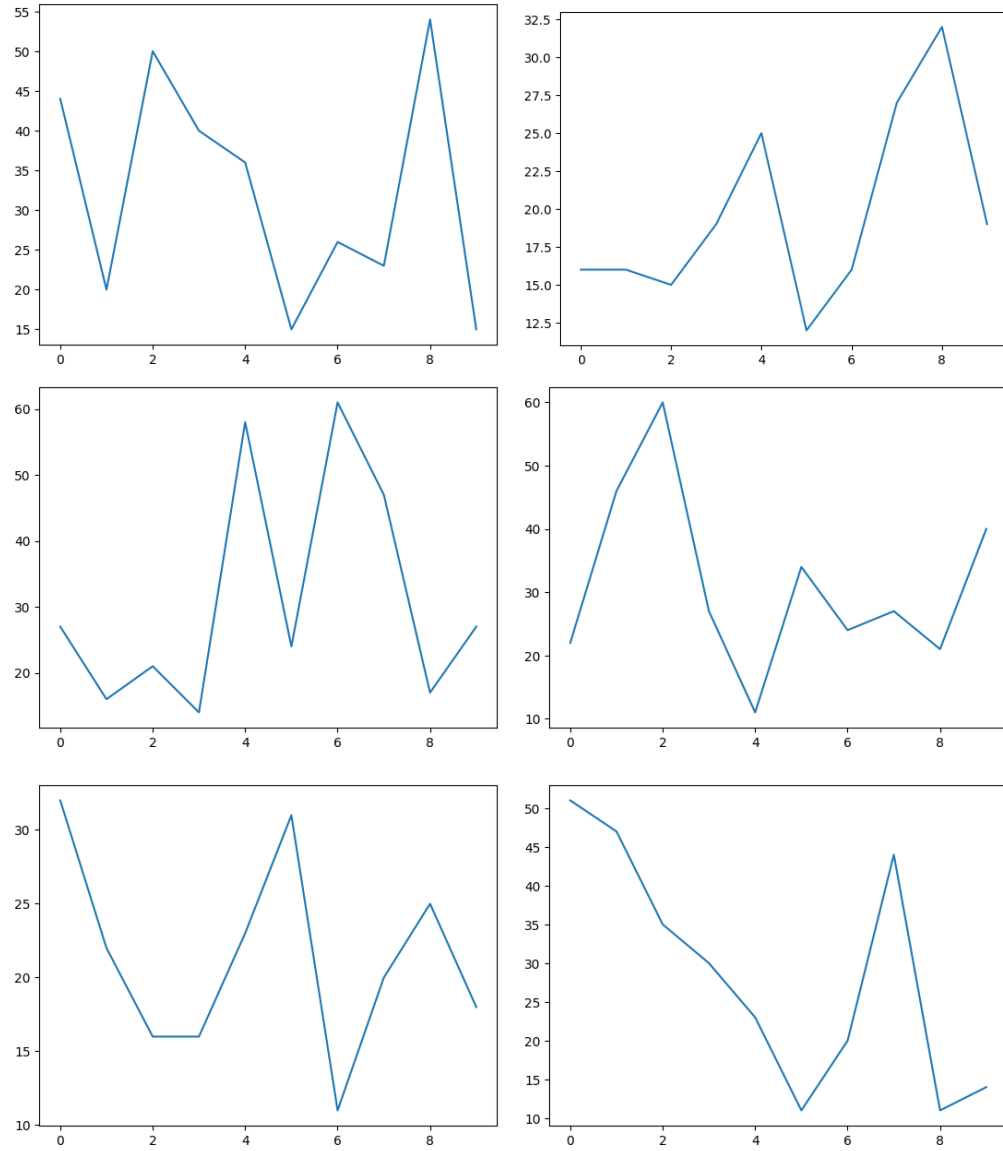


Figura 6.3.: Cartpole: Evaluación del Generador GAIL en las épocas 50, 60, 70, 80, 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)

Finalmente, mostramos el valor medio de cada evaluación del generador para así poder extraer conclusiones sobre el aprendizaje ya que las gráficas anteriores solo me muestran que el agente persigue un comportamiento robusto pero para saber si está aprendiendo, necesitamos valorar la tendencia de las recompensas medias por época de evaluación del generador.

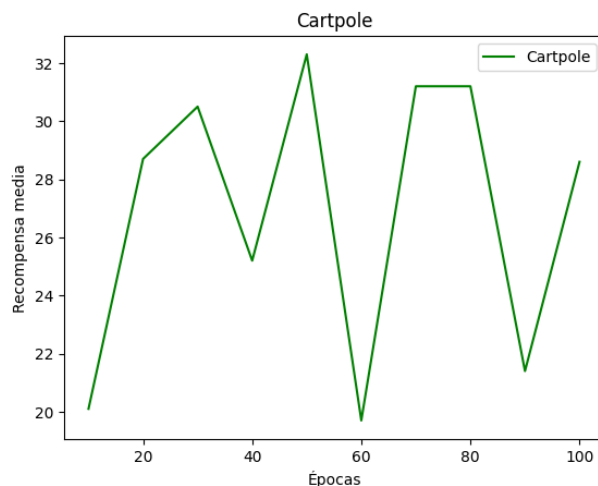


Figura 6.4.: Evaluación del Generador GAIL Cartpole: épocas-Recompensa media

Con el entorno CartPole se obtienen unas recompensas de media oscilantes, consiguiendo que cuando se llegue a la época 100 y se finalice el entrenamiento su recompensa de media sea superior a la inicial. En general, todas son superiores a la inicial excepto la obtenida en la época 60. Por tanto, se está realizando un entrenamiento muy bueno. De hecho, se consigue un incremento de 8 unidades en la recompensa obtenida al final del entrenamiento, en la época 100 con respecto a la evaluación de la política en la época 10. En definitiva, hemos conseguido ajustar muy bien la técnica GAIL para el entorno CartPole.

6.4. Taxi v3

El entorno de Taxi es parte de los entornos de Toy Text del artículo [Die00]. Tenemos un espacio de acciones discreto de tamaño 6 y un espacio de observaciones de tamaño 500. En este entorno tenemos cuatro ubicaciones designadas en el mundo de la cuadrícula indicadas por R(ojo), V(erde), A(marillo) y A(zul). Cuando comienza el episodio, el taxi comienza en una cuadrícula aleatoria y el pasajero está en una ubicación aleatoria. El taxi se dirige a la ubicación del pasajero, recoge al pasajero, se dirige al destino del pasajero (otra de las cuatro ubicaciones especificadas) y luego deja al pasajero. Una vez que el pasajero es dejado, el episodio termina. En cuanto a las acciones y observaciones tenemos que especificar:

- Hay 6 acciones discretas deterministas:
 - 0: mover hacia el sur
 - 1: mover hacia el norte
 - 2: mover hacia el este
 - 3: mover hacia el oeste
 - 4: recoger al pasajero
 - 5: dejar al pasajero
- Observaciones: Hay 500 estados discretos posibles ya que hay 25 posiciones para el taxi, 5 posibles ubicaciones del pasajero (incluido el caso en que el pasajero está en el taxi) y 4 ubicaciones posibles de destino.
- Ubicaciones de pasajeros: 0: R(ojo), 1: V(erde) , 2: A(marillo), 3: A(zul), 4: en el taxi
- Destinos: 0: R(ojo), 1: V(erde), 2: A(marillo), 3: A(zul)

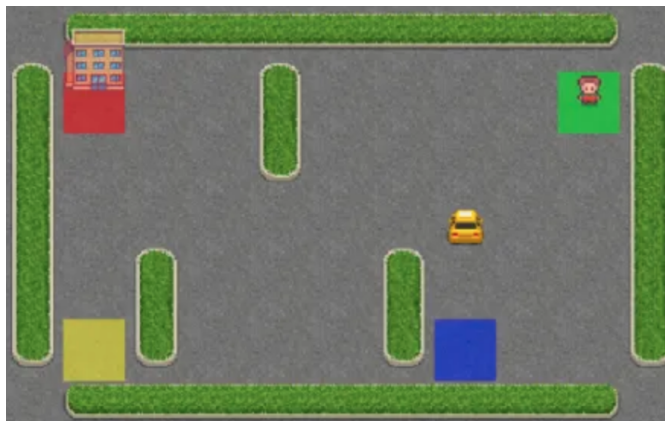


Figura 6.5.: Entorno de Toy Text: Taxi [TTK⁺23]

6.4.1. Configuración

Definiremos $N = 2500$, entonces existirán 2500 matrices Q y tendremos un tensor de tamaño $(500, 6, 2500)$. Estos datos se dividen en entrenamiento y validación, de forma que el 80 % son para entrenar y el 20 % para evaluar. En resumen, tendremos 2000 tablas Q para entrenar y 500 para evaluar.

Decimos que es una base de datos real porque ejecutaremos el algoritmo Q-Learning para obtener la tabla Q real, entonces la tabla Q^* sintética tratará de imitar a Q con objeto de que su política sea similar a la política obtenida con el algoritmo Q-Learning.

6.4.2. Ejecución

En primer lugar, estudiamos la técnica HQL sobre el entorno Taxi. Se realiza un entrenamiento con 100 épocas, utilizando el optimizador Adam con una tasa de aprendizaje de 0.0001 con valores de β_1 a 0.5, β_2 a 0.9. Al igual que hicimos con el entorno CartPole, cada 10 épocas de entrenamiento evaluamos al generador para determinar las recompensas de las tablas Q^* sintéticas que se esta construyendo y así tener una medida de como de buena es la política que se está generando.

En segundo lugar, se va a aplicar la técnica GAIL al entorno Taxi para ello se realizarán algunos cambios respecto a la configuración GAIL que se implemento para CartPole con objeto de mejorar sus resultados. Estas modificaciones se deben a que estamos trabajando con un entorno donde las observaciones son discretas, entonces la implementación de GAIL necesitaba algunos pequeños cambios.

Se implementa un discriminador mas sofisticado ya que debe recordar el dato discreto de la observación del entorno perteneciente al intervalo $[1, 500]$. Para ello, la arquitectura de la red neuronal del discriminador utiliza dos capas de Memoria a Largo Plazo de Corto Plazo (LSTM) antes de las capas densas. Asimismo, se tuvo que modificar la entrada a las redes neuronales de GAIL ya que ahora el tamaño de la observación es solo 1.

El proceso de entrenamiento se realiza a lo largo de 100 épocas con un optimizador Adam que utiliza la misma tasa de aprendizaje, 0.0001, e igual que se hizo con la técnica HQL se evaluará la política que se está aprendiendo cada 10 épocas, evaluando al generador a lo largo de 10 episodios y en la tabla vemos la media de los 10 valores de recompensa.

6.4.3. Resultados

Experimento 1: HQL

Analizamos la Tabla 6.3 con el objetivo de extraer conclusiones basadas en los datos obtenidos en este experimento.

Podemos ver como las pérdidas del discriminador disminuyen aunque sus valores sean negativos mientras que las pérdidas del generador oscilan ya que primero comienzan a disminuir y después aumentan. Respecto a la recompensa de media obtenida en la evaluación de 100 episodios nuevos cada 10 épocas de entrenamiento, se observa un aumento del valor obtenido en la época 10 al valor de la época 100. Aunque existan fluctuaciones

en los valores del resto de épocas, tenemos que tener en cuenta que finalmente el agente actúa con una política mas similar a la experta ya que las recompensa de media aumenta a un valor de 8.46 cuando termina el proceso de entrenamiento.

Epochs	Loss D	Loss G	Reward Mean
1	-0.0025	-0.3446	-
10	-0.0137	-0.3769	7.88
20	-0.0266	-0.4608	7.65
30	-0.0390	-0.5417	7.59
40	-0.0509	-0.3751	7.82
50	-0.0605	-0.3481	7.68
60	-0.0797	-0.5493	7.92
70	-0.0916	-0.5414	8.24
80	-0.1124	-0.4064	7.63
90	-0.1292	-0.4262	7.85
100	-0.1451	-0.3129	8.46

Tabla 6.3.: Taxi: Entrenamiento HQL

Seguidamente, evaluamos el proceso de entrenamiento del generador y del discriminador, obteniendo los siguientes valores de la función de pérdida en Tabla 6.4. La pérdida del discriminador en los datos reales es inferior a la pérdida en los datos falsos, lo cual es completamente esperado.

Loss real	Loss fakes
-0.54101	-0.68708

Tabla 6.4.: Taxi: Evaluación Discriminador HQL

En segundo lugar, mostramos las gráficas obtenidas en la evaluación del generador a lo largo del proceso de entrenamiento. Notemos que la última gráfica esta evaluando al generador una vez ha terminado su proceso de entrenamiento a lo largo de las 100 épocas.

Las gráficas aparecen ordenadas de izquierda a derecha y de arriba a abajo.

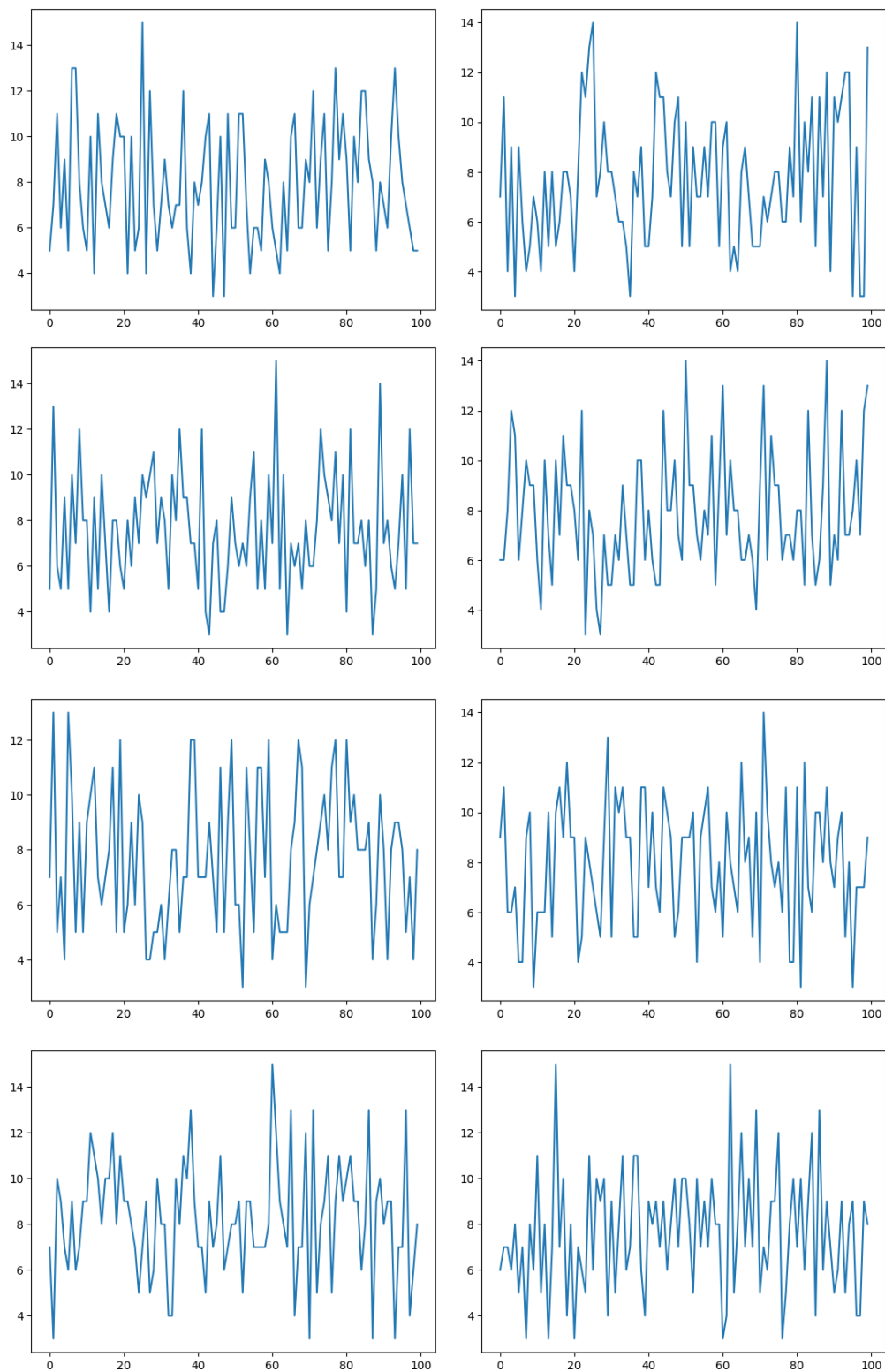


Figura 6.6.: Taxi: Evaluación del Generador HQL en las épocas 10, 20, 30, 40, 50, 60, 70, 80 (Eje x-Nº episodio, Eje y-Recompensa)

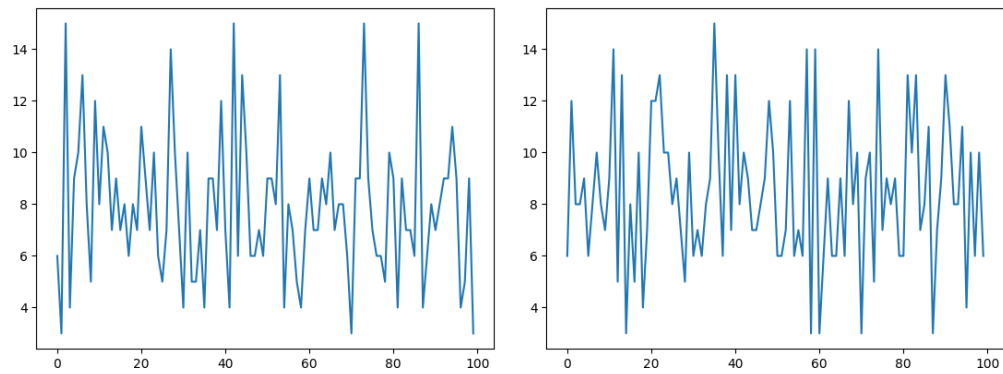


Figura 6.7.: Taxi: Evaluación del Generador HQL en las épocas 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)

A continuación, mostramos en una gráfica los valores de recompensa medio obtenido por cada 10 valores de recompensas en cada evaluación. En esta gráfica se puede visualizar como las recompensas van aumentando conforme avanza el entrenamiento del modelo. Entonces, cuando termina el proceso de entrenamiento, tenemos a un agente que interactúa con el entorno produciendo recompensas superiores.

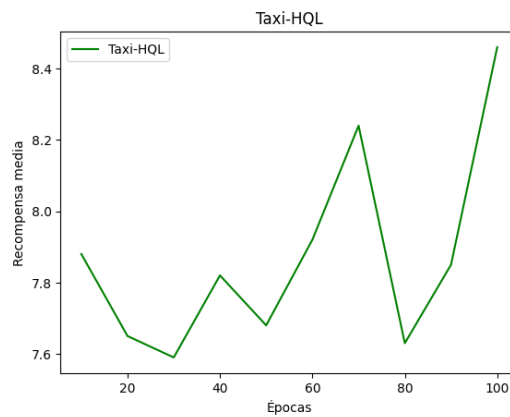


Figura 6.8.: Evaluación del Generador HQL Taxi: épocas-Recompensa Media

Experimento 2: GAIL

Primero, visualizamos Tabla 6.5 con las pérdidas del generador y del discriminador junto con las recompensas medias para tener un análisis inicial del experimento al igual que hicimos anteriormente.

Podemos apreciar como la pérdida del discriminador disminuye considerablemente, esto es, cada vez distingue mejor las secuencias de estados-acciones verdaderas de las secuencias sintéticas. La pérdida del generador oscila y cuando evaluamos su comportamiento ocurre lo mismo.

Es un experimento costoso ya que el dominio del espacio de observaciones es bastante amplio. Como trabajo futuro, se podría plantear estudiar como mejorar este experimento porque si comparamos la recompensa media de la época 10 con el valor obtenido en la época 100, existe un descenso en vez de un incremento.

Epochs	Loss D	Loss G	Reward Mean
1	1.1155	9.9856	-
10	1.0030	10.0590	-648.0
20	0.8804	1.0593	-745.4
30	0.6916	1.0586	-752.6
40	0.7136	1.0590	-767.0
50	0.5806	1.0586	-759.5
60	0.5114	10.0676	-705.0
70	0.3754	1.0670	-696.5
80	0.4209	10.0575	-731.0
90	0.3859	1.0718	-781.4
100	0.3877	10.0927	-738.2

Tabla 6.5.: Taxi: Entrenamiento GAIL

A continuación, se procede a evaluar el comportamiento del Discriminador que como se ha ilustrado en Tabla 6.6 es capaz de aprender muy rápido así que se obtiene valores muy bajos tanto para las secuencias de observaciones-acciones verdaderas como para las sintéticas, siendo inferior para las primeras que para las segundas.

Loss real	Loss fakes
0.02334	0.03390

Tabla 6.6.: Taxi: Evaluación Discriminador GAIL

Finalmente, nos queda evaluar la capacidad de nuestro Generador de GAIL. Como se ha ido evaluando al generador en el entrenamiento de cada 10 épocas, en la última evaluación, esto es, cuando evaluamos al generador en la época 100 se obtiene la evaluación del comportamiento del generador una vez ha finalizado el proceso de entrenamiento.

Así tenemos las siguientes gráficas de las recompensas obtenidas en cada evaluación realizada a lo largo de 10 episodios, se encuentran ordenadas de izquierda a derecha y de arriba a abajo.

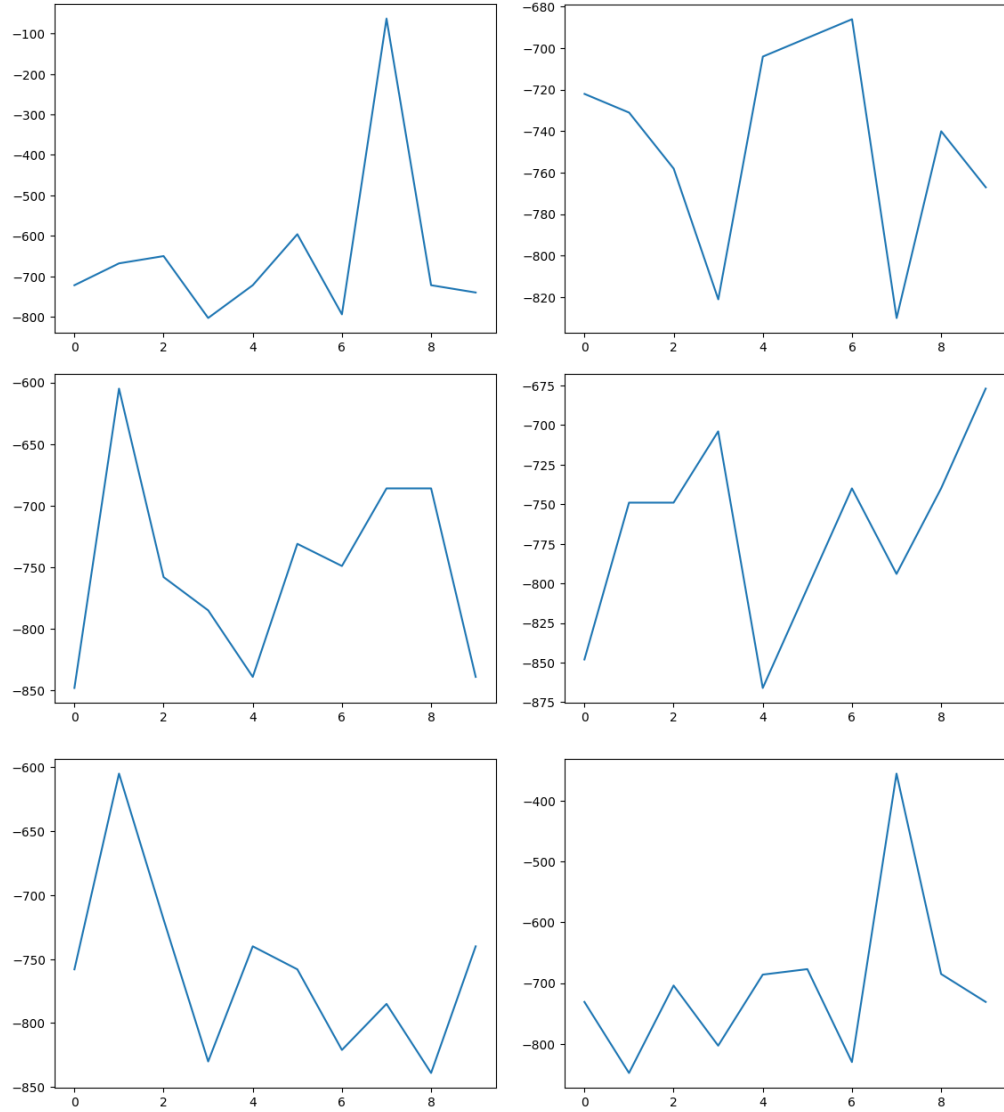


Figura 6.9.: Taxi: Evaluación del Generador GAIL en las épocas 10, 20, 30, 40, 50, 60 (Eje x-Nº episodio, Eje y-Recompensa)

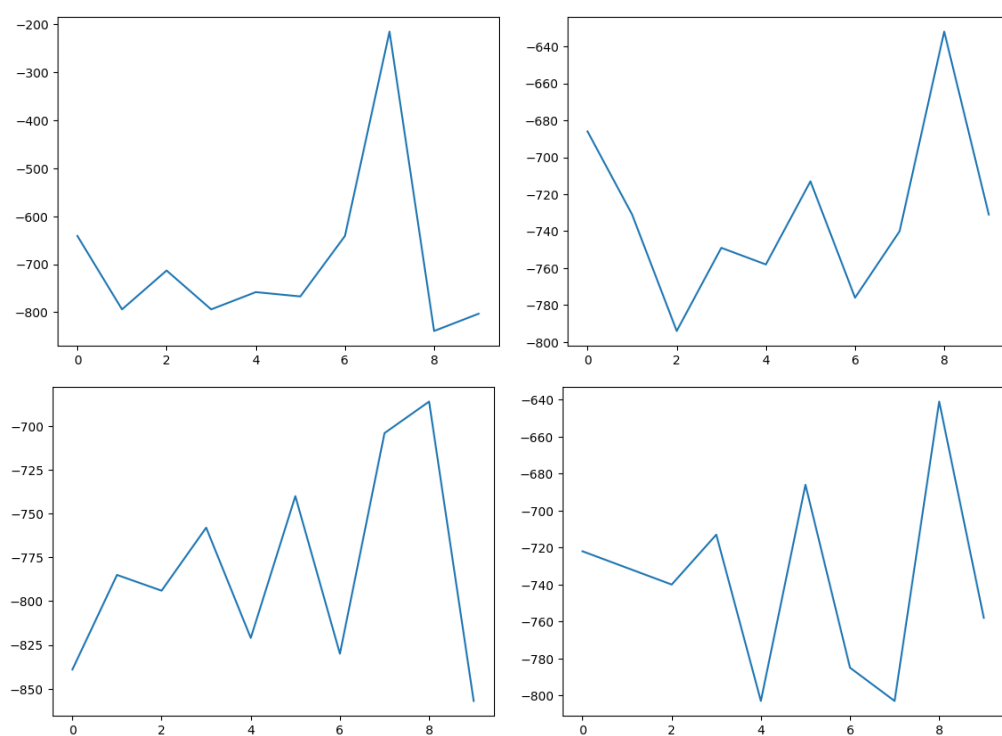


Figura 6.10.: Taxi: Evaluación del Generador GAIL en las épocas 70, 80, 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)

Observamos que al evaluar el comportamiento de nuestro agente mientras se va entrenando con GAIL, las recompensas medias van decrementando. Un comportamiento óptimo sería un aumento lineal de la recompensa por el número de épocas. Es cierto, que se producen aumentos en algunas épocas pero comparando el resultado de la época 100 con el de la época 10, existe un decremento cuando termina el proceso de entrenamiento.

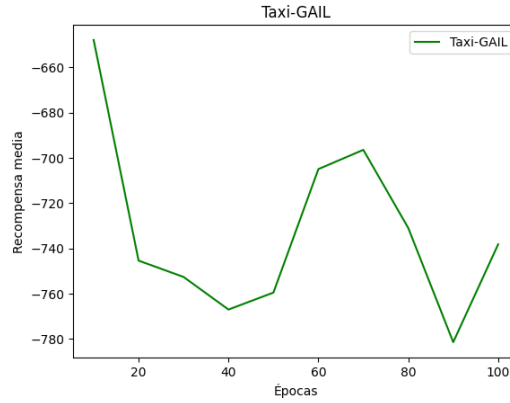


Figura 6.11.: Evaluación del Generador GAIL Taxi: épocas-Recompensa Media

Finalmente, como HQL y GAIL son técnicas de aprendizaje por imitación que se están aplicando sobre el entorno Taxi, se ha decidido estudiar las analogías entre sus comportamientos sobre dicho entorno en Sección 8.1.

7. Experimentación con entornos de Sinergym

En este capítulo trataremos de probar GAIL con un entorno encargado de controlar los sistemas HVAC de distintos edificios, llamado Sinergym. Los sistemas HVAC son sistemas de calefacción (**H**eating), ventilación (**V**entilation) y aire acondicionado (**A**ir **C**onditioning) que se instalan en edificios inteligentes. Estos sistemas pretenden realizar su misión consiguiendo dos objetivos principales: el ahorro energético y el confort de los residentes. Debido a que se trata de entornos con acciones y estados continuos, solo realizamos la experimentación con GAIL en los escenarios seleccionados (5Zone, Datacenter y Warehouse). Los códigos de las implementaciones de los distintos experimentos podemos estudiarlo en https://github.com/silviabm98/GAIL_Sinergym

7.1. Entorno de ejecución

A continuación, se va a mencionar las herramientas utilizadas en nuestra experimentación.

- Esta experimentación se ha llevado a cabo en un ordenador ASUS TUF Gaming F15 FX507ZM-TUF507ZM que presenta las siguientes características:
 - Sistema operativo: Ubuntu 22
 - Procesador: 12th Gen Intel(R) Core(TM) i7-12700H
 - GPU: Nvidia Geforce RTX 3060
- La herramienta Sinergym permite crear entornos siguiendo la interfaz de Gymnasium, con el propósito de encapsular los motores de simulación diseñados para el control de edificios mediante el enfoque del aprendizaje profundo por refuerzo.

Esta herramienta contiene funcionalidades muy diversas: incluye diferentes motores de simulación estableciendo una comunicación entre el lenguaje de programación Python y EnergyPlus; define entornos con diferentes edificios, climas, espacios de acción/observación, funciones de recompensa, etc; también tiene entornos personalizables donde el usuario puede definir su propio modelo combinando diferentes edificios, climas, recompensas, espacios de observación/acción, etc o usar alguno predefinido y cambiar algún aspecto; contiene componentes personalizables, como puede ser crear nuevas funciones de recompensas, wrappers, controladores, etc; esta

herramienta proporciona funcionalidad para obtener información sobre los entornos, como las zonas o los planificadores disponibles en el modelo de edificio; en último lugar, Sinergym presenta una adaptación automática del modelo de edificio a los cambios del usuario.

- Por otra parte, en estos experimentos se han utilizado todas las bibliotecas y herramientas de Python comentadas en la sección 6.1.

7.2. Descripción

Tenemos un espacio de estados continuo $\mathcal{S}$ y un conjunto de observaciones discretas $\mathcal{A}$, podemos estudiar políticas del conjunto $\Pi = \{\pi : \mathcal{S} \rightarrow \mathcal{A}\}$. De esta forma, dado cualquier estado $S_i \in \mathcal{S}$, $\forall i\{1, 2, \dots\}$, esto es, cualquier variable que nos aporte información útil para determinar los valores que establece el sistema HVAC del edificio, nuestro agente de aprendizaje por refuerzo sabe siguiendo una política que acción de $\mathcal{A} = \{a_1, a_2, \dots\}$ realizar para cuando tomamos un valor en ese estado.

Esta política se puede conseguir con algoritmos que aprendan gradientes de políticas parametrizadas y en consecuencia, obtengan la política óptima dado un espacio de estados y acciones con un entorno donde el agente va modificando su estado al realizar una acción tras otra.

Este trabajo de investigación persigue aplicar modelos generativos al aprendizaje profundo por refuerzo, por tanto, se quiere generar nuevas políticas. En otras palabras, se pretende generar nuevas secuencias de estados y acciones que definen nuevas políticas aprendidas.

Básicamente, la teoría de GAIL nos demuestra como podemos perfectamente aplicar una GAN con objeto de aprender una política π^* lo más similar posible a la política experta π_E pero sin tener acceso a ella.

En nuestra experimentación, se trata de aplicar la teoría de GAIL donde aprenderemos una política, esto es, una función $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$, donde el espacio de estados y acciones dependerá del edificio que escojamos. En la Sección 5.3 se han nombrado los estados y las acciones así que simplemente se trata de encontrar un mapeo entre estados y acciones que se parezca a la política experta, $\pi_E : \mathcal{S} \rightarrow \mathcal{A}$, $\pi^* \sim \pi$

Para construir nuestra base de datos experta, recopilaremos información sobre los estados y las acciones correspondientes en dos archivos CSV. Luego, dividiremos esta base de datos en dos conjuntos: uno para entrenamiento y otro para evaluación.

Inicialmente, crearemos un archivo CSV para las observaciones y otro para las acciones correspondientes donde se almacenarán los datos referentes a dos años. Estas observaciones y acciones se corresponden con las obtenidas por una política experta, política que selecciona la acción para cada observación del entorno utilizando un modelo de aprendizaje por refuerzo preentrenado. En este caso, se ha utilizado un PPO preentrenado con un total de 25000 pasos. Finalmente, utilizaremos un 20 % de estos estados con sus acciones correspondientes para evaluar nuestros experimentos con GAIL.

Los objetivos del aprendizaje por imitación los tenemos claro, trataremos de aprender una política parametrizada que imite a la política experta donde los estados-acciones

son los correspondientes a los sistemas HVAC de los edificios.

¿Entonces, qué contenido estamos generando con esta experimentación de GAIL? Estamos generando continuamente estados y acciones sintéticas, esto es, estados-acciones correspondientes al sistema HVAC del edificio y de las cuales extraeremos la política parametrizada.

Finalmente, un agente inteligente controlará los sistemas HVAC de dichos edificios a partir de las secuencias de estados-acciones sintéticas ya que su comportamiento será similar al comportamiento experto y esto se conseguirá sin necesidad de realizar numerosas interacciones de prueba y error entre el agente y el entorno.

Obsérvese que tenemos un espacio de observaciones continuo así que no hemos podido probar la técnica de Imitación Q-Learning sobre los edificio del entorno Sinergym.

7.3. 5Zone

7.3.1. Configuración

La política experta viene definida en dos archivos CSV, uno de observaciones y otro de acciones. En 5Zone tendremos 17 tipo de observaciones continuas y 10 acciones discretas en 5Zone. Concatenando los datos de estos archivos, realizando un preprocesamiento de las acciones a partir de una codificación uno-en-caliente (OneHotEncoder), construimos la secuencia experta $[s_i, a_i] \forall i$.

$$[s_i, a_i] = \{s_{i1}, s_{i2}, s_{i3}, \dots, s_{i17}, a_{i1}, a_{i2}, \dots, a_{i10}\} \forall i$$

A partir de esta base de datos experta, ya podemos realizar nuestra división en conjunto de entrenamiento (80 %) y validación (20 %) para comenzar a entrenar nuestro modelo GAIL. Así, tendremos 28032 instancias $[s_i, a_i]$ para el entrenamiento y 7008 para la evaluación.

7.3.2. Ejecución

Se realiza un entrenamiento a lo largo de 100 épocas haciendo uso de un optimizador Adam con una tasa de aprendizaje de $1e - 4$. Cada 10 épocas de entrenamiento se evalúa el comportamiento del generador para ver la capacidad que tiene nuestro agente de producir secuencias de estados-acciones e interactuar con el entorno a partir de estas secuencias sintéticas.

7.3.3. Resultados

A lo largo del entrenamiento analizamos los datos almacenados en Tabla 7.1, la pérdida del Discriminador va disminuyendo, algo que es esperable ya que el Discriminador cada vez va aprendiendo conforme va recibiendo entradas sintéticas y verdaderas al incrementar las épocas en el proceso de entrenamiento. Cada vez se le hace más fácil discriminar entre $[s, a]$ y $[s, a]^*$. La pérdida del generador, esto es, la pérdida de PPO, va oscilando entorno al cero con valores bajos.

Si nos fijamos en los valores de las recompensas medias obtenidas al ir evaluando al generador cada 10 épocas, observamos como se producen oscilaciones pero si comparamos el valor de la primera época con la última existe un aumento de la recompensa. Esto quiere decir, que el comportamiento de la política final es mejor que el de la política inicial al evaluarse sobre 5 episodios un agente sobre este entorno. Se ha reducido el número de episodios en cada evaluación del generador porque un episodio en Sinergym contiene datos almacenado de 2 años.

Epochs	Loss D	Loss G	Reward Mean
1	6.3705	0.1131	
10	6.3570	0.0982	-43646.9
20	6.1808	0.0951	-43625.9
30	6.1337	0.5180	-43621.8
40	5.8999	0.7637	-43647.3
50	5.2638	0.0949	-43618.7
60	4.1636	0.0951	-43595.8
70	3.7802	0.1406	-43645.1
80	3.6494	0.1370	-43627.4
90	3.5500	0.0951	-43636.5
100	3.3727	0.0951	-43601.01

Tabla 7.1.: 5Zone: Entrenamiento GAIL

Procedemos a evaluar el discriminador del entrenamiento de nuestra GAIL en este experimento, teniendo en cuenta que las pérdidas del discriminador superan a 1. Para un entorno así de complejo, se obtienen valores de pérdidas notables como podemos ver en Tabla 7.2.

Loss real	Loss fakes
1.09331	1.47892

Tabla 7.2.: 5Zone: Evaluación Discriminador

Finalmente, vemos las gráficas de los valores de recompensas que dan de media los valores de la tabla de entrenamiento. Las gráficas se ordenan de izquierda a derecha y de arriba a abajo, esto es, la gráfica de la primera fila a la izquierda se corresponde con la evaluación del generador en la época 10 y la de su derecha se corresponde con la evaluación en la época 20.

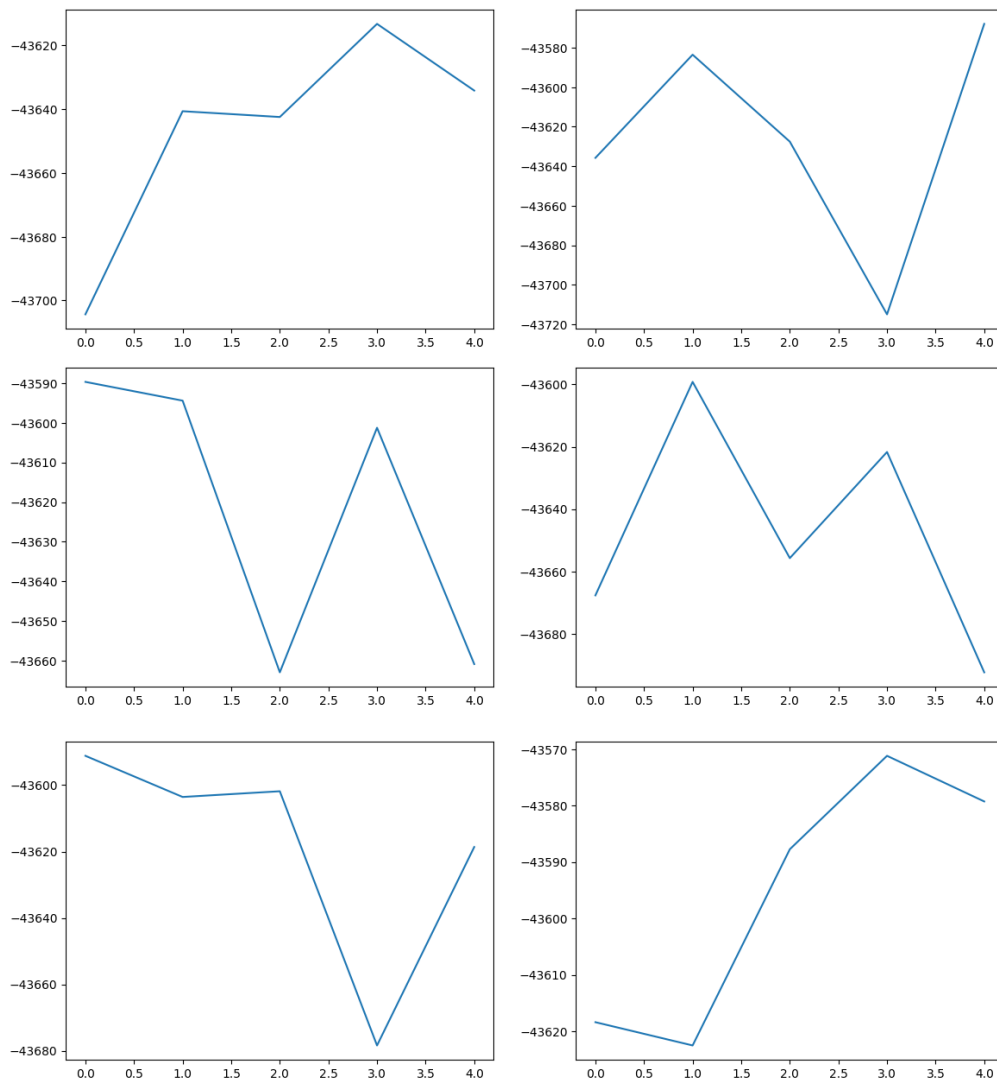


Figura 7.1.: 5Zone: Evaluación del Generador en las épocas 10, 20, 30, 40, 50, 60 (Eje x-Nº episodio, Eje y-Recompensa)

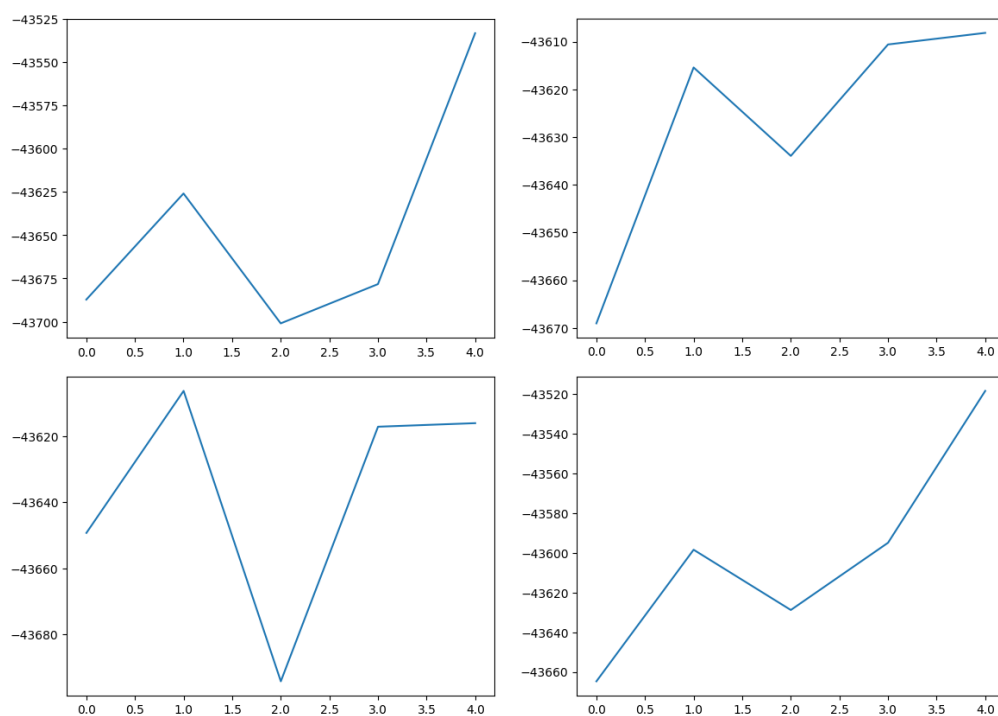


Figura 7.2.: 5Zone: Evaluación del Generador en las épocas 70, 80, 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)

Se analiza el comportamiento de GAIL en entorno 5Zone donde podemos visualizar en la siguiente gráfica las recompensas de media de cada evaluación del generador anterior. Se ha verificado que GAIL ha aprendido en este proceso de entrenamiento ya que conseguimos que la recompensa de la época 100, esto es, la recompensa obtenida cuando finaliza el entrenamiento supere a la recompensa inicial al evaluar el agente sobre dicho entorno.

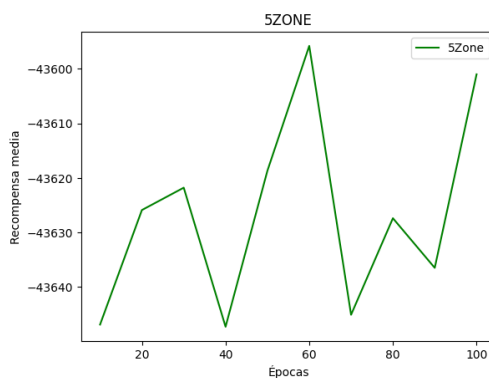


Figura 7.3.: Evaluación del Generador GAIL 5Zone: épocas-Recompensa Media

7.4. Datacenter

7.4.1. Configuración

En este entorno tenemos 29 tipo de observaciones continuas y 10 acciones discretas. Así que, el archivo `observations_DataCenter.csv` tendrá entradas de 29 valores donde la entrada i se corresponde con $s_i = \{s_{i1}, s_{i2}, s_{i3}, \dots, s_{i29}\}$ y en `observations_DataCenter.csv` cada entrada tendrá un único valor correspondiente con $a_i \in \{0, 2, 3, \dots, 9\}$.

Concatenando los datos de estos archivos, realizando un preprocesamiento de las acciones a partir de una codificación uno-en-caliente (OneHotEncoder), construimos la secuencia experta $[s_i, a_i] \forall i$.

$$[s_i, a_i] = \{s_{i1}, s_{i2}, s_{i3}, \dots, s_{i29}, a_{i1}, a_{i2}, \dots, a_{i10}\} \forall i$$

A partir de esta base de datos experta, ya podemos realizar nuestra división en conjunto de entrenamiento (80 %) y validación (20 %) para comenzar a entrenar nuestro modelo GAIL. Así, tendremos 28032 instancias $[s_i, a_i]$ para el entrenamiento y 7008 para la evaluación.

7.4.2. Ejecución

Se realiza un entrenamiento a lo largo de 100 épocas aplicando el optimizador Adam con un valor de tasa de aprendizaje de $1e - 4$, de β_1 de 0.5 y de ϵ de $1e - 8$. Cada 10 épocas de entrenamiento se evalúa el comportamiento del generador para ver la capacidad que tiene nuestro agente de producir secuencias de estados-acciones e interactuar con el entorno a partir de estas secuencias sintéticas.

7.4.3. Resultados

La pérdida del discriminador podemos apreciar en Tabla 7.3 como oscila y finalmente, aumenta muy levemente, mientras que la pérdida del generador tiene oscilaciones pero finalmente aumenta también ligeramente.

Epochs	Loss D	Loss G	Reward Mean
1	2.0526	1.5162	
10	2.0542	1.5150	-75631.07
20	2.0537	1.5064	-75621.6
30	2.0546	1.8174	-75641.4
40	2.0529	1.4967	-75605
50	2.0518	1.1100	-75643.09
60	2.0512	1.4957	-75633.5
70	2.0550	1.49681	-75633.8
80	2.0532	1.5249	-75632.5
90	2.0517	1.5796	-75622.7
100	2.0550	1.5222	-75672.2

Tabla 7.3.: Datacenter: Entrenamiento GAIL

Cuando evaluamos el Discriminador, nos dimos cuenta en Tabla 7.4 que este comportamiento dio muy buenos resultados dando lugar a valores de funciones de pérdida inferiores a 1 en datos reales, $[s, a]$, y datos falsos, $[s, a]^*$.

Loss real	Loss fakes
0.00197	$3.79394328453575e - 11$

Tabla 7.4.: Datacenter: Evaluación Discriminador

Seguidamente, podemos ver las gráficas obtenidas al evaluar al generador cada 10 épocas de entrenamiento. Se encuentran ordenadas al igual que en 5Zone, de izquierda a derecha y de arriba a abajo.

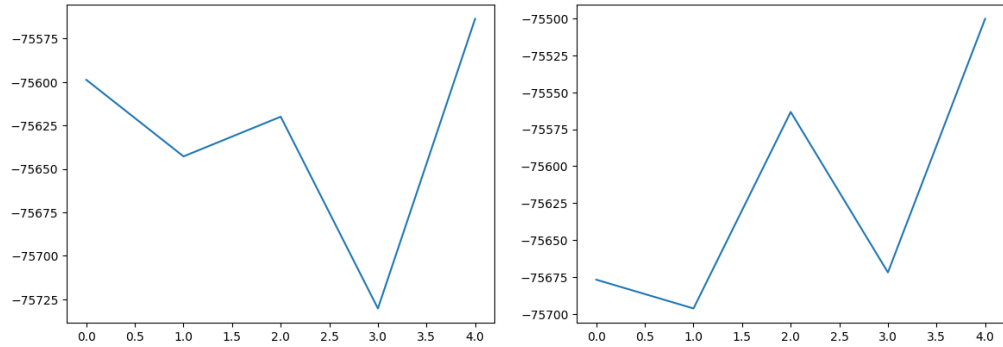


Figura 7.4.: Datacenter: Evaluación del Generador en las épocas: 10, 20 (Eje x-Nº episodio, Eje y-Recompensa)

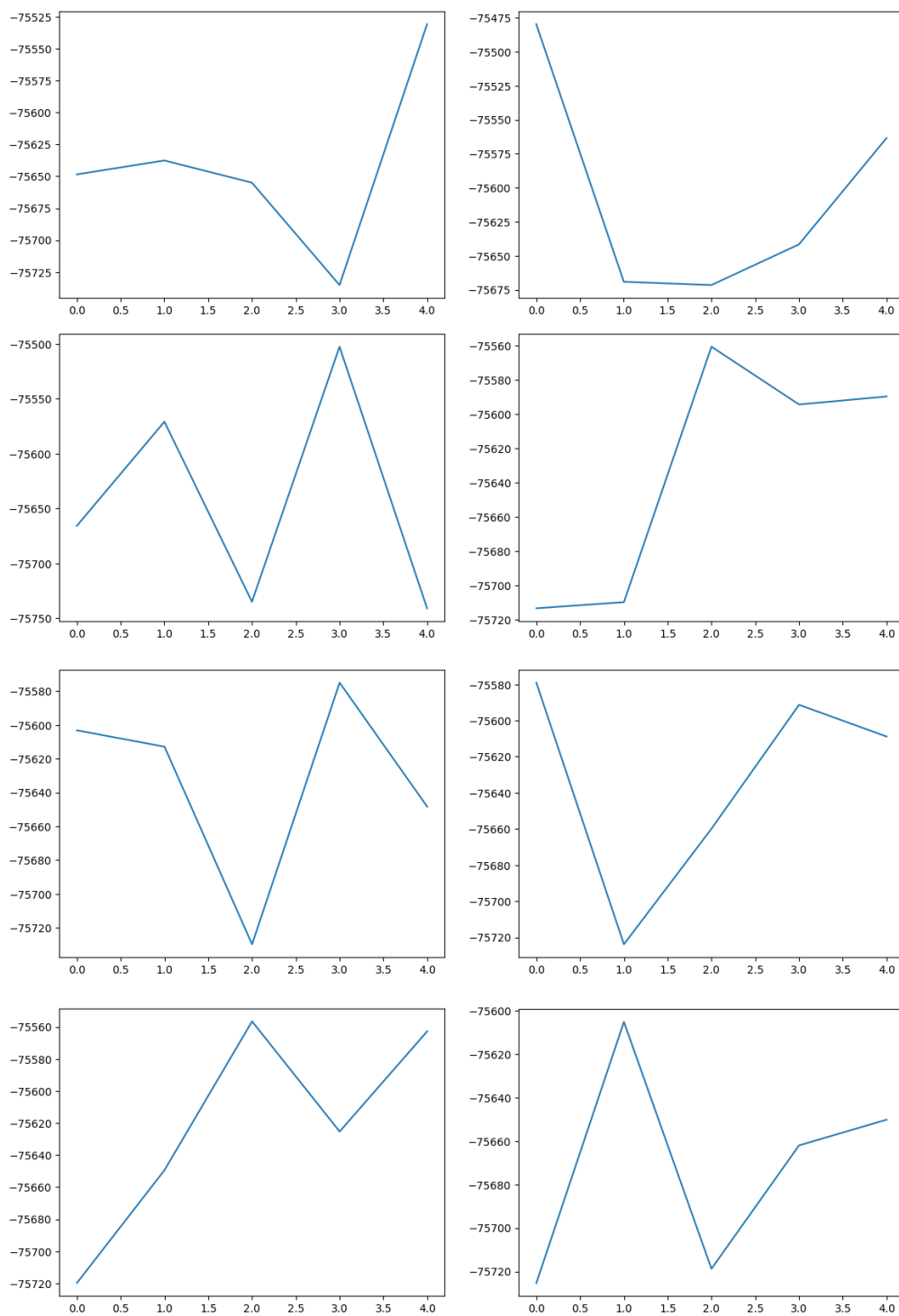


Figura 7.5.: Datacenter: Evaluación del Generador en las épocas: 30, 40, 50, 60, 70, 80, 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)

A continuación, vemos como se presenta un comportamiento aceptable hasta la época 90 pero en la época 100 disminuye muy fuertemente. Lo peor de este entorno es que sus recompensas se alejaban mucho de cero. Sin embargo, aunque el generador no obtenga el mejor comportamiento, conseguimos que el Discriminador no logre distinguir la política experta de la política generada. La política experta se obtiene de un modelo PPO actuando sobre el entorno, quizás le cueste al modelo PPO actuar sobre dicho entorno ya que se trata de un entorno algo mas complejo.

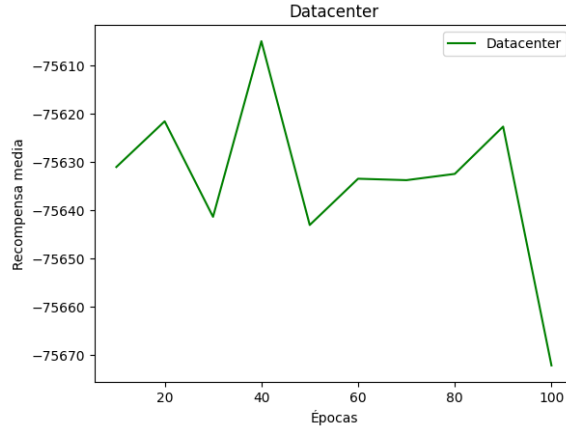


Figura 7.6.: Comparativa Datacenter

7.5. Warehouse

7.5.1. Configuración

En el entorno Warehouse tenemos 23 tipo de observaciones continuas y 10 acciones discretas. Si concatenamos los datos de los archivos correspondientes a este entorno al igual que hicimos con 5Zone y Datacenter, realizando un preprocesamiento de las acciones a partir de una codificación uno-en-caliente (OneHotEncoder), construimos la secuencia experta $[s_i, a_i] \forall i$.

$$[s_i, a_i] = \{s_{i1}, s_{i2}, s_{i3}, \dots, s_{i23}, a_{i1}, a_{i2}, \dots, a_{i10}\} \forall i$$

A partir de esta base de datos experta, ya podemos realizar nuestra división en conjunto de entrenamiento (80 %) y validación (20 %) para comenzar a entrenar nuestro modelo GAIL. Así, tendremos 28032 instancias $[s_i, a_i]$ para el entrenamiento y 7008 para la evaluación.

7.5.2. Ejecución

Realizamos un entrenamiento a lo largo de 100 épocas donde volvemos a utilizar el optimizador Adam con una tasa de aprendizaje de $1e - 04$. Cada 10 épocas de entrenamiento se evalúa el comportamiento del generador para ver la capacidad que tiene

nuestro agente de producir secuencias de estados-acciones e interactuar con el entorno a partir de estas secuencias sintéticas.

7.5.3. Resultados

Vemos en Tabla 7.5 como la pérdida del discriminador se decrementa muy levemente mientras el generador oscila con valores entorno al -0.3 .

Los valores de las recompensas medias oscilan, produciéndose un decremento del valor en la época 100 con respecto al obtenido en la época 10.

Epochs	Loss D	Loss G	Reward Mean
1	6.6567	-0.3157	
10	6.6564	-0.3144	-45995.8
20	6.6565	-0.2957	-45984.0
30	6.6590	-0.2987	-46006.1
40	6.6584	-0.2971	-45982.3
50	6.6573	-0.3206	-45966.1
60	6.6572	-0.3054	-45970.1
70	6.6580	-0.2938	-45976.10
80	6.6585	-0.2921	-45994.6
90	6.6593	-0.3096	-46000.7
100	6.6563	-0.3336	-46001.9

Tabla 7.5.: Warehouse: Entrenamiento GAIL

Después de entrenar a nuestro modelo, vamos a evaluar al generador y al discriminador. Podemos apreciar en Tabla 7.6 como las pérdidas reales son inferiores a las pérdidas falsas de la función de pérdida del Discriminador cuando se evalúa.

Loss real	Loss fakes
4.605164	4.605165

Tabla 7.6.: Warehouse: Evaluación Discriminador

Al tener valores de pérdidas del discriminador entorno a 4, no estamos obteniendo un comportamiento tan bueno como cuando obteníamos valores inferiores o alrededor de 1 y esto se ve reflejado cuando tratamos de evaluar al generador, intentado ver como actuaría un agente sobre nuestro entorno con dichas secuencias sintéticas $[s, a]^*$. Por ende, se ve plasmado en la tabla en los valores de recompensa media donde en el valor de la época 100 es inferior al de la época 10.

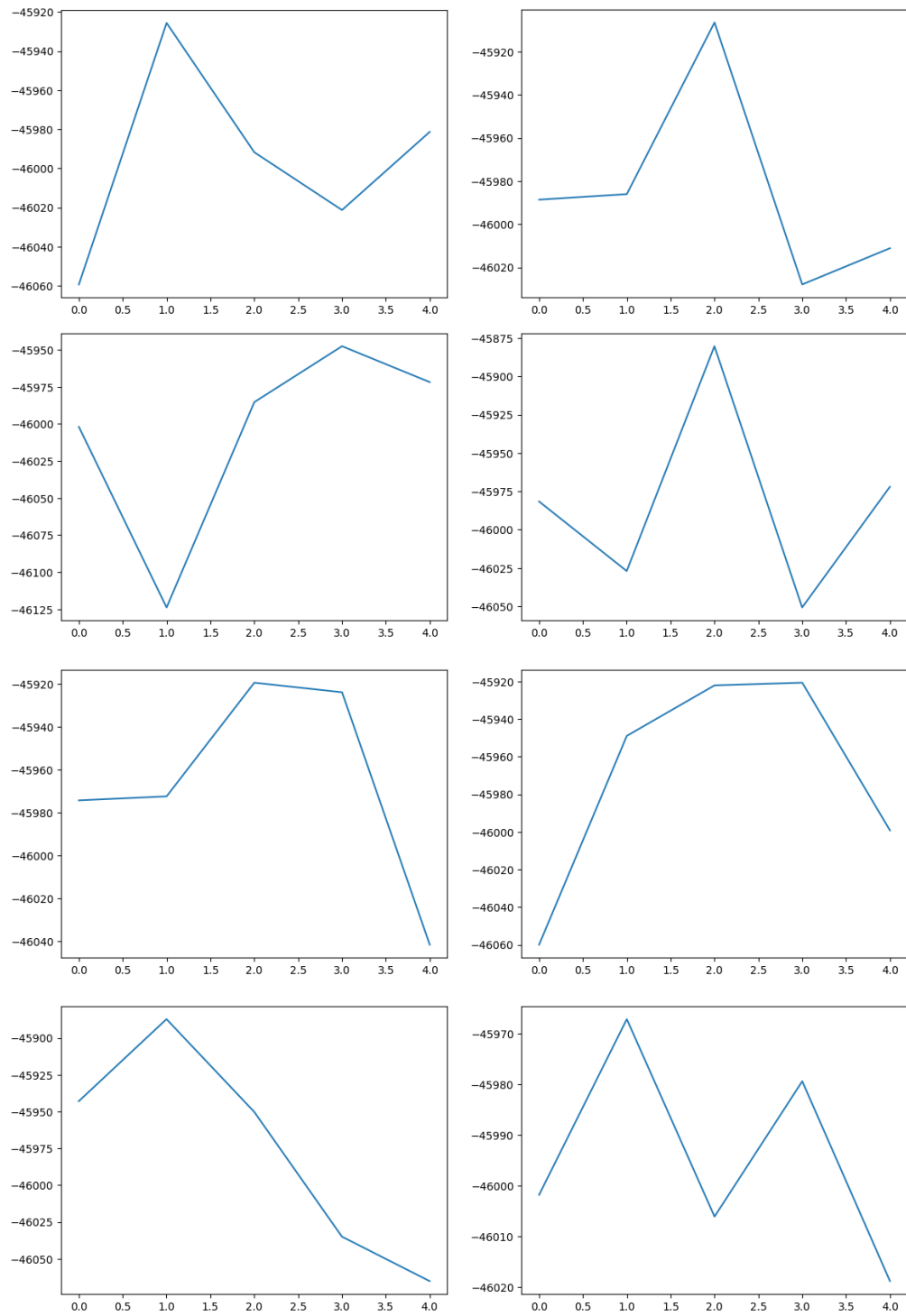


Figura 7.7.: Warehouse: Evaluación del Generador en las épocas: 10, 20, 30, 40, 50, 60, 70, 80 (Eje x-Nº episodio, Eje y-Recompensa)

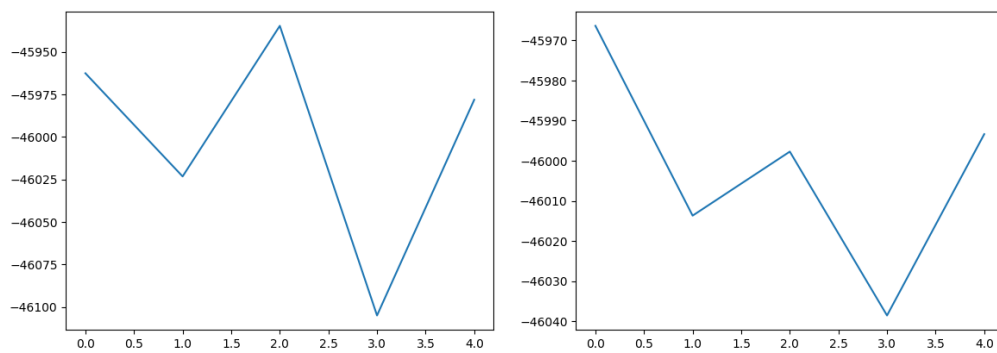


Figura 7.8.: Warehouse: Evaluación del Generador en las épocas: 90, 100 (Eje x-Nº episodio, Eje y-Recompensa)

Comparando el comportamiento de GAIL sobre Warehouse con respecto a los anteriores entornos, apreciamos que las recompensas son mucho menores que en Datacenter y toman valores mas similares a los de 5Zone. Si comparamos la actuación de la técnica GAIL sobre 5Zone y sobre este entorno, ganaría 5Zone ya que en este entorno las recompensas empiezan a decrecer a partir de la época 50. De hecho, la recompensa obtenida cuando finaliza el entrenamiento es inferior a la obtenida en la época 10 y esto no sucedía en 5Zone.

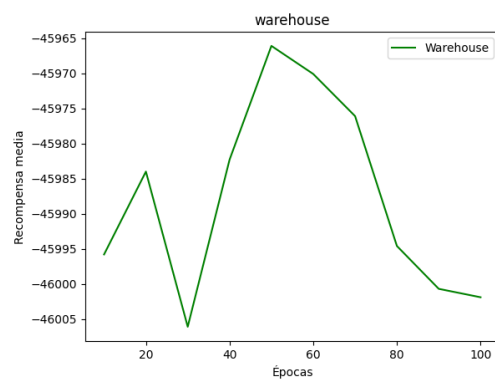


Figura 7.9.: Comparativa Warehouse

8. Discusión

En este capítulo, se procede a comparar en tablas y en gráficas los resultados obtenidos a lo largo del entrenamiento y evaluación de nuestras técnicas sobre los distintos entornos. Discutiremos sobre si es mejor GAIL o HQL para el entorno de Taxi, y compararemos los resultados obtenidos con GAIL con respecto a los obtenidos con un algoritmo de aprendizaje profundo por refuerzo, PPO.

8.0.1. Resumen de resultados

La Tabla 8.1 engloba las recompensas medias que teníamos en las tablas de entrenamiento de cada uno de los entornos con objeto de poder visualizarlas todas juntas y comparar el comportamiento del agente sobre cada entorno aplicando la técnica GAIL o HQL.

La última fila de las tablas de entrenamiento de cada entorno nos muestra la recompensa de media obtenida cuando el proceso de entrenamiento ha finalizado ya que todos los entornos se entrenan a lo largo de 100 épocas. Por este motivo, para comparar nos basaremos tanto en este valor como en las fluctuaciones existentes en el resto de épocas y en la diferencia con respecto a la época 10.

A todo esto, está claro es que nos interesa que en un entorno las recompensas medias sean lo mas grande posible así que esto se priorizará a la hora de comparar una técnica frente a otra.

	GAIL	GAIL	GAIL	GAIL	GAIL	HQL
Épocas	5ZONE	DATACENTER	WAREHOUSE	CARTPOLE	TAXI	TAXI
10	-43646.9	-75631.07	-45995.8	20.1	-648.0	7.88
20	-43625.9	-75621.6	-45984.0	28.7	-745.4	7.65
30	-43621.8	-75641.4	-46006.1	30.5	-752.6	7.59
40	-43647.3	-75605	-45982.3	25.2	-767.0	7.82
50	-43618.7	-75643.09	-45966.1	32.3	-759.5	7.68
60	-43595.8	-75633.5	-45970.1	19.7	-705.0	7.92
70	-43645.1	-75633.8	-45976.10	31.2	-696.5	8.24
80	-43627.4	-75632.5	-45994.6	31.2	-731.0	7.63
90	-43636.5	-75622.7	-46000.7	21.4	-781.4	7.85
100	-43601.01	-75672.2	-46001.9	28.6	-738.2	8.46

Tabla 8.1.: Comparativa global: Recompensas medias obtenidas cuando se evalúa el comportamiento del agente en cada época de entrenamiento

8.1. Comparativa HQL vs GAIL en Taxi

El entorno Taxi se ha implementado con las dos técnicas planteadas de aprendizaje por imitación. A continuación, observamos que al evaluar el comportamiento de nuestro agente mientras se va entrenando con GAIL, las recompensas de media van decreciendo. Un comportamiento óptimo sería un aumento lineal de la recompensa por el número de épocas. Es cierto, que se producen aumentos en algunas épocas pero comparando el resultado de al época 100 con el de la época 10, existe un decremento cuando termina el proceso de entrenamiento.

Por ello, se decide probar con la técnica HQL donde se puede visualizar como las recompensas van aumentando conforme avanza el entrenamiento del modelo. Entonces, cuando termina el proceso de entrenamiento, tenemos a un agente que interactúa con el entorno produciendo recompensas superiores.

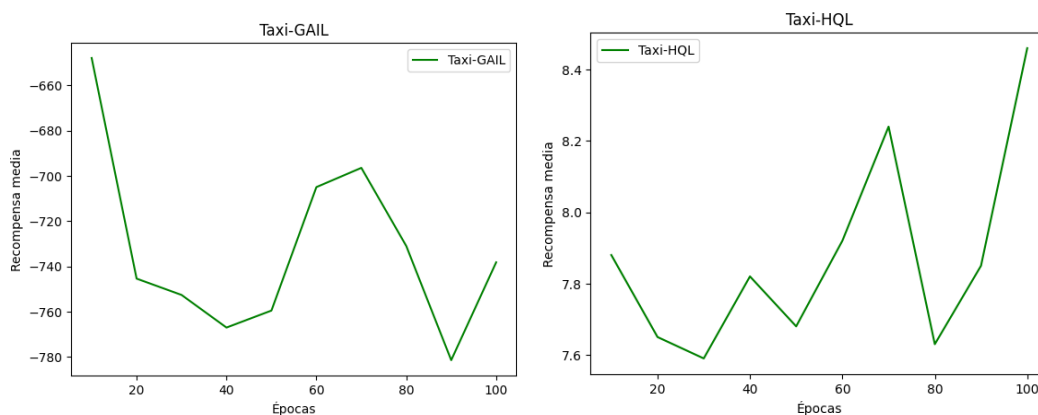


Figura 8.1.: Comparativa Taxi: GAIL vs HQL

En vista de los resultados obtenidos al evaluar un agente sobre el entorno Taxi con secuencias de observaciones-acciones falsas, tenemos que el comportamiento de HQL supera a GAIL. Esto se debe, a que la curva de recompensas producida al evaluar al agente sobre el entorno toma valores negativos y con tendencias locales decrecientes en GAIL mientras que en HQL predominan los valores positivos y las tendencias crecientes.

Comprobamos la actuación del modelo PPO sobre este entorno devolviéndonos un valor de recompensa medio de -200 . El siguiente gráfico de barra nos ilustra las tres técnicas.

En entornos de OpenAI Gym, se ha verificado que mi propuesta da resultados razonables para el entorno Taxi, superando a GAIL y a un PPO implementado para un total de 25 pasos.

8.2. Comparativa GAIL vs PPO

En último lugar, vamos a realizar una comparativa de las recompensas producidas al evaluar el comportamiento de la política generada con GAIL y las recompensas obtenidas

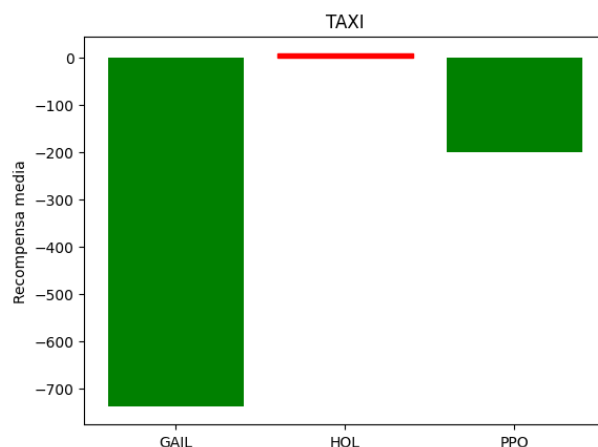


Figura 8.2.: Comparativa Taxi: GAIL vs HQL vs PPO

al evaluar la política obtenida con el algoritmo PPO. De esta forma, podemos comparar como de buena ha sido la actuación de GAIL sobre cada uno de los edificios. Claramente, el algoritmo PPO obtendrá recompensas superiores ya que se trata de un algoritmo de aprendizaje mas simple que el aprendizaje por imitación generativo adversario (GAIL), entonces en un tiempo t obtendrá PPO unas acciones que devolverán recompensas superiores que las recompensas correspondientes a las acciones seleccionadas por GAIL. Esta comparativa nos sirve para cuantificar cuanto se aproxima GAIL a PPO y así demostrar que nuestro algoritmo GAIL puede aproximarse a obtener políticas tan buenas como las calculadas con el algoritmo de PPO.

■ **5Zone:**

- Con GAIL podemos visualizar en las gráficas de 7.1 y 7.2 como las recompensas se encuentran aproximadamente en el intervalo $[-43720, -43520]$, obteniendo en la última época un promedio de media en las recompensas de -43601 .
- Con PPO obtenemos una recompensa promedia de -42662 , es un valor mas bajo que los anteriores y mas cercano a cero. Sin embargo, GAIL aunque no consiga ser tan bueno como PPO pero alcanza una muy buena aproximación ya que sus recompensas medias solo difieren aproximadamente en 936.

■ **Datacenter:**

- Con GAIL podemos visualizar en las gráficas de 7.4 y de 7.5 como las recompensas se distribuyen a lo largo del intervalo $[-75750, -75475]$, obteniendo en la última época una recompensa media de -75672.2 .
- Con PPO obtenemos una recompensa promedia -48925.43 , así que nuestra actuación con GAIL para el entorno Datacenter no resulta tan buena como para los entornos Warehouse y 5Zone. También, tenemos que tener en cuenta

que Datacenter es un entorno con un número superior de observaciones que 5Zone y nuestro modelo le esta costando aprender exclusivamente de datos limitados sin acceso al entorno, con PPO es mucho mas fácil ya que aprende a partir de prueba y error, de acceso continuo al entorno.

Tenemos que tener en cuenta que PPO no es algoritmo de aprendizaje por imitación, es un algoritmo de aprendizaje por refuerzo profundo donde tiene acceso al entorno en cada paso de un episodio.

■ **Warehouse**

- Con GAIL podemos visualizar en las gráficas del experimento 7.7 y de 7.8 como los valores de recompensas que se obtienen se encuentran en el intervalo $[-46125, -45875]$, obteniendo en la última época una recompensa de media de 46001.9
- Con PPO obtenemos una recompensa promedia de 44725.37, entonces nuestra implementación solo difiere de PPO 1276.5 unidades.

Comparando los resultados obtenidos con los alcanzados con PPO, podemos afirmar que se ha logrado buenas aproximaciones con los edificios 5Zone y con Warehouse ya que las recompensas de media difieren muy poco con las de PPO. El entrenamiento de GAIL bajo las configuraciones establecidas para estos entornos nos ha proporcionado muy buenos modelos.

No obstante, el entorno Datacenter no se ajusta tan bien, como línea futura estaría bien analizar como mejorar este proceso de entrenamiento con objetivo de obtener un mejor modelo.

Parte III.

Conclusiones y trabajo futuro

9. Conclusiones y Trabajo Futuro

En este trabajo fin de máster se ha estudiado la conexión tan útil que podemos establecer entre los modelos generativos y el aprendizaje por refuerzo. A día de hoy, la aplicación de modelos generativos es cada vez más común. Estos modelos permiten generar información muy diversa y el aprendizaje por imitación se aprovecha de ello, utilizando las redes generativas adversarias y presentando su enfoque de aprendizaje por imitación generativo adversario (GAIL).

Un agente de aprendizaje por imitación aprende a actuar en un entorno sin necesidad de haber interactuado directamente con él, aprende exclusivamente de datos limitados. Esto es potencialmente útil ya que nuestro agente esta aprendiendo a saber actuar en un entorno sin interactuar previamente a base de prueba y error.

Mas bien, la técnica GAIL perteneciente al aprendizaje por imitación, a partir de una secuencia finita de estados-acciones, construye un generador capaz de proporcionar todas las secuencias de estados-acciones falsas que le solicitemos y además, estas secuencias falsas son muy parecidas a las reales ya que el discriminador le cuesta tratar de distinguirlas.

Por tanto, cada vez que queramos que nuestro agente se mueva por un entorno sin necesidad de interactuar a base de experimentos de prueba y error, le solicitamos a nuestro generador una secuencia de estados-acciones falsas, y ya tendremos a nuestro agente experimentando correctamente en el entorno que deseemos. Esto se ha probado con los entornos CartPole y Taxi de Gym OpenAI y con tres edificios del entorno Sinergym, se ha demostrado las altas capacidades que tiene un generador de GAIL una vez tengamos a nuestro modelo entrenado. Con Sinergym se ha puesto en marcha GAIL sobre un entorno que controla los sistemas HVAC de simulaciones de edificios, lo que ha supuesto cierta complejidad adicional con respecto al entorno CartPole o Taxi.

Cumpliendo este propósito de generar secuencias falsas, se está generando una política falsa que trata de imitar a la política experta. Por ese motivo, GAIL pertenece a la familia de algoritmos de aprendizaje por imitación porque se esta tratando de imitar a la política que sigue la secuencia de estados-acciones expertas. Esto es, para un estado del entorno el experto realizará tal acción pues justo ese comportamiento se está tratando de imitar y replicar en la política falsa de forma que el discriminador dude a la hora de distinguir si el comportamiento es de un experto o es falso.

Nos cuestionamos que modelo generativo podríamos usar ya que existen muchos en la literatura. Aquí se ha presentado la propuesta de GAIL, que básicamente es una modificación de las redes generativas adversarias (GANs), pero se podría intentar plantear como línea futura ver que ocurre si intentamos cambiar este modelo generativo.

Por otro lado, se ha presentado una nueva propuesta dentro del marco de la hibridación entre el aprendizaje por imitación y el aprendizaje por refuerzo denominada Hibridación Q-Learning (HQL). En esta propuesta, surge el interrogante acerca de la viabilidad de emplear una red generativa adversaria para la creación de tablas Q-learning con el propósito de generar políticas haciendo uso del algoritmo Q-Learning similares a las política expertas establecida por las tablas Q de la base de datos experta. En resumen, esta propuesta abre un amplio abanico de posibilidades de desarrollo, que aunque se encuentra en sus primeras etapas, ya ha sido objeto de experimentación en el entorno del problema del Taxi, arrojando resultados notables.

Se ha verificado como una GAN puede generar soluciones tabulares del algoritmo de control *off-policy* Q-Learning, tablas falsas notadas como Q^* . Y como de esas soluciones se puede extraer su política π que tratará de imitar a la política experta π_E de la base de datos experta formada por tablas Q-Learning, notadas como Q .

Como línea futura, se plantea cambiar el modelo generativo de las GANs e intentar explorar una nueva conexión con el aprendizaje por refuerzo. A continuación, se analizará una introducción de los modelos Decision Difusser con objeto de entusiasmar al lector. Además, quedaría pendiente utilizar nuestras implementaciones con nuevos entornos, así como tratar de mejorar nuestros experimentos.

A. Uso de Decision Diffuser en Aprendizaje por refuerzo

Para la generación de contenido sintético con redes neuronales vamos a estudiar también otras aproximaciones de tipo "generativo", como los modelos Decision Diffuser donde podemos aplicar aprendizaje por refuerzo y desarrollar un línea de investigación similar sustituyendo las GANs por dicho modelo. Para ello, se ha analizado en gran detalle la documentación [ADG⁺23].

A.1. Introducción

En los últimos años han habido avances increíbles en el campo de los modelos generativos condicionales: modelos de imágenes como Dall-E ó ImageGen, modelos de lenguaje como GPT ó Minerva.

En la literatura, se ha demostrado que ha tenido un gran éxito el problema de la toma de decisiones desde la perspectiva de un modelo generativo condicional, accediendo a un conjunto de datos fuera de línea (offline). Por tanto, se prioriza trabajar el problema de la toma de decisiones fuera de línea (offline) que principalmente se ha basado en herramientas de aprendizaje por refuerzo.

En la siguiente imagen vemos una idea del problema de toma de decisiones usando modelos generativos condicionales.

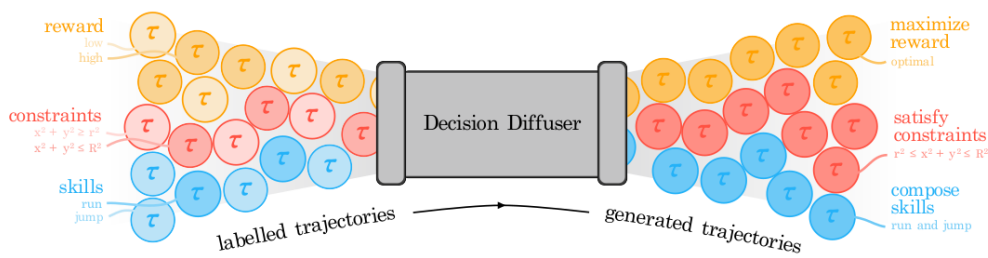


Figura A.1.: A partir de unas secuencias de trayectorias etiquetadas, generamos unas trayectorias que cumplen: maximizan la señal de recompensas, satisfacen restricciones y establecen mejoras [ADG⁺23]

A.1.1. Aprendizaje por refuerzo offline

Vamos a formular un problema de toma de decisiones como un proceso de decisión de Markov definido por la tupla $(\rho_0, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, donde ρ_0 es la distribución del estado inicial, $\mathcal{S}$ y $\mathcal{A}$ son los espacios de estados y acciones, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ es la función de transición, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ nos da la recompensa de cualquier transición y $\gamma \in [0, 1)$ es un factor de descuento.

El agente actúa con la política estocástica $\pi : \mathcal{S} \rightarrow \Delta_{\mathcal{A}}$ generando una secuencia de transiciones estado-acción-recompensa, trayectorias $\tau := (s_k, a_k, r_k)_{k \geq 0}$ con probabilidad $p_{\pi}(\tau)$ y devolviendo la recompensa $R(\tau) := \sum_{k \geq 0} \gamma^k r_k$

- El objetivo en aprendizaje por refuerzo es encontrar la política que maximiza la recompensa devuelta, $\pi^* = \arg \max_{\pi} \mathbb{E}_{\tau \sim P_{\pi}} [R(\tau)]$
- Nuestro objetivo en **aprendizaje por refuerzo fuera de línea (offline)**, encontrar la política que maximiza la recompensa devuelta de un conjunto de datos de transiciones fijado y coleccionados gracias a una política μ desconocida. Utilizando el aprendizaje de diferencia temporal, TD, conseguiremos que la distribución de probabilidad de los estados visitados, $d^{\pi_{\phi}}$, se aproxime a d^{μ} . Los algoritmos de RL offline imponen una restricción en la distancia entre estas dos distribuciones de probabilidad, $D(d^{\pi_{\phi}} || d^{\mu})$.

A.1.2. Modelos de difusión probabilísticos

Definición A.1. *Los **modelos de difusión** son un específico tipo de modelo generativo que aprenden la distribución de probabilidad $q(x)$ de los datos de un conjunto de datos $\mathcal{D} := \{x^i\}_{0 \leq i \leq M}$.*

- *El procedimiento de generación de datos es modelado con un proceso de ruido predefinido a partir de una distribución Normal (distribución gaussiana):*

$$q(x_{k+1}|x_k) := \mathcal{N}(x_{k+1}; \sqrt{\alpha_k}x_k, (1 - \alpha))$$

- *Proceso de entrenamiento inverso, θ son los pesos del entrenamiento:*

$$p_{\theta}(x_{k-1}|x_k) := \mathcal{N}(x_{k-1}|\mu_{\theta}(x_k, k), \Sigma_k)$$

donde $\mathcal{N}(\mu, \Sigma)$ denota una distribución normal con media μ y varianza Σ , $\alpha_k \in \mathbb{R}$ determina un programa de varianza, $x_0 := x$ es una muestra, $x_1, x_2, \dots, x_{K-1}$ es el espacio latente y $x_K \sim \mathcal{N}(0, I)$ para una elección cuidadosa de un α_k y de un valor grande de K .

Aunque el límite inferior variacional pueda manejarse con la función de pérdida de $\log p_{\theta}$ para optimizarse en el entrenamiento de modelos de difusión, se propone simplificar y sustituir la función de pérdida por

$$\mathcal{L}(\theta) := \mathbb{E}_{k \sim [1, K], x_0 \sim q, \epsilon \sim \mathcal{N}(0, I)} [||\epsilon - \epsilon_{\theta}(x_k, k)||^2]$$

El ruido predicho $\epsilon_\theta(x_k, k)$ es parametrizado con una red neuronal profunda, estimando $\epsilon \sim \mathcal{N}(0, I)$ al agregar una muestra x_0 a la base de datos para producir el ruido x_k . Este proceso es similar a predecir la media de $p_\theta(x_{k-1}|x_k)$ desde $\mu_\theta(x_k, k)$ para ser calculada cómo una función de $\epsilon_\theta(x_k, k)$.

Difusión guiada o libre

La equivalencia que existe entre los modelos de difusión y igualación de puntajes ("score-matching"), guía a dos tipos de métodos de condicionamiento: clasificación guiada y la clasificación libre.

- Clasificación guiada: $\epsilon_\theta(x_k, y, k)$
- Clasificación libre: Representamos el ruido incondicional $\epsilon_\theta(x_k, k)$, como $\epsilon_\theta(x_k, \emptyset, k)$, donde $\emptyset$ toma el lugar de la etiqueta y

Un modelo **Decision Diffuser** utiliza clasificación libre para guiar sus decisiones, la siguiente imagen ilustra y explica su proceso.

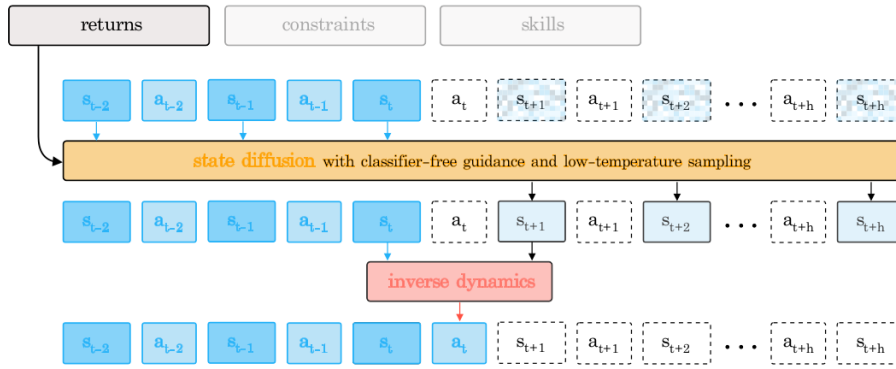


Figura A.2.: **Planificación en Decision Diffuser:** Dado un estado s_t y sus condiciones, el modelo Decision Diffuser utiliza clasificación libre. A partir de s_t , generamos una secuencia de futuros estados $\{s_{t+1}, s_{t+2}, \dots, s_{t+h}\}$ y utilizamos la dinámica inversa para extraer y ejecutar la acción a_t que guía a los estados futuros [ADG⁺23]

A.2. Modelado generativo con Decision Diffuser

Formulamos una toma de decisiones secuencial como un problema estándar de modelado generativo condicional:

$$\max_{\theta} \mathbb{E}_{\tau \sim \mathcal{D}} [\log p_{\theta}(x_0(\tau) | y(\tau))]$$

Nuestro objetivo es estimar una distribución de datos condicional con p_θ así podemos generar porciones de una trayectorias $x_0(\tau)$ de información sobre $y(\tau)$. La construcción de nuestro modelo generativo se realiza de acuerdo a las condiciones del proceso de difusión condicionales:

- $q(x_{k+1}(\tau)|x_k(\tau))$
- $p_\theta(x_{k-1}(\tau)|x_k(\tau), y(\tau))$

A.2.1. Estados y acciones

Como en la naturaleza del aprendizaje por refuerzo, la mayoría de las observaciones son continuas, mientras que las acciones a veces son discretas y otras veces continuas, se decide difundir únicamente las observaciones.

$$x_k(\tau) := (s_t, s_{t+1}, \dots, s_{t+H-1})_k$$

El valor k nota el paso de tiempo (timestep) del proceso y t el tiempo dentro de la trayectoria τ , H es la longitud de la trayectoria.

Dinámica inversa

Como vimos anteriormente, el modelo dinámico inverso te permite obtener acciones. Veamos con la siguiente fórmula, como aprendiendo una función f_ϕ , podemos obtener acciones a partir de dos estados consecutivos.

$$a_t := f_\phi(s_t, s_{t+1})$$

Básicamente, estamos prediciendo acciones con un modelo dinámico inverso.

A.2.2. Planificación con orientación un clasificador libre

Entrenamos un clasificador $p_\phi(y(\tau)|x_k(\tau))$ para predecir $y(\tau)$ a partir de las trayectorias ruidosas $x_k(\tau)$.

Para implementar un clasificador libre, nos tomamos una muestra $x_0(\tau)$ que es el inicio de un ruido Gaussiano $x_K(\tau)$ y vamos refinando $x_k(\tau)$ en $x_{k-1}(\tau)$ en cada paso intermedio con la siguiente perturbación del ruido:

$$\bar{\epsilon} := \epsilon_\theta(x_k(\tau), \emptyset, k) + \omega(\epsilon_\theta(x_k(\tau), y(\tau), k) - \epsilon_\theta(x_k(\tau), \emptyset, k))$$

Primero, nosotros observamos el estado y el entorno, muestreamos y finalmente, identificamos la acción. Este proceso se describirá mas detalladamente en el algoritmo 15

A.2.3. Condicionamiento mas allá del retorno

No estamos definiendo condiciones en la variable $y(\tau)$, nosotros podríamos también guiar nuestras secuencias de estados para que satisfagan ciertas restricciones. Un ejemplo de introducir una restricción sería imponer que se maximice la recompensa de las acciones que se escojan.

Para generar trayectorias que maximicen la recompensa devuelta en cada acción, condicionamos el ruido del modelo de una trayectoria en el valor devuelto de la siguiente forma:

$$\epsilon_\theta(x_k(\tau), y(\tau), k) = \epsilon_\theta(x_k(\tau), R(\tau), k)$$

donde $R(\tau) \in [0, 1]$

Algorithm 15: Planificación condicional con Decision Diffuser

Entrada: ϵ_θ modelo de ruido, f_ϕ dinámica inversa, ω escala guiada, C longitud de la historia, y condición.

1. Iniciamos $h \rightarrow \text{Queue}(\text{length} = C), t \rightarrow 0$
 2. **Mientras** $s \neq$ estado final **hacer**
 - a) Observar el estado s
 - b) $h.\text{insert}(s)$
 - c) Inicializar $x_K(\tau) \sim \mathcal{N}(0, \sigma I)$
 - d) **Para** $k = K \dots 1$ **hacer**
 - 1) $x_k(\tau)[:\text{length}(h)] \leftarrow h$
 - 2) $\bar{\epsilon} \leftarrow \epsilon_\theta(x_k(\tau), \emptyset, k) + \omega(\epsilon_\theta(x_k(\tau), y(\tau), k) - \epsilon_\theta(x_k(\tau), \emptyset, k))$
 - 3) $(\mu_{k-1}, \sum_{k-1}) \rightarrow \text{Denoise}(x_k(\tau), \bar{\epsilon})$
 - 4) $x_{k-1} \sim \mathcal{N}(\mu_{k-1}, \alpha \sum_{k-1})$
 - e) **fin**
 - f) Extraer (s_t, s_{t+1}) de $x_0(\tau)$
 - g) Ejecutar $a_t = f_\phi(s_t, s_{t+1})$
 - h) $t \rightarrow t + 1$
 3. **fin**
-

A.2.4. Entrenamiento de Decision Diffuser

Dado un dataset de trayectorias $\mathcal{D}$, cada etiqueta con la recompensa que se alcanza, la restricción que se satisface, ó la habilidad que se demuestra, entonces entrenamos al proceso de difusión p_θ parametrizándolo a través del modelo de ruido ϵ_θ y el modelo inverso dinámico f_ϕ a través de la siguiente función de pérdida. Esta función la trataremos de minimizar para ambos parámetros: θ y ϕ .

Función de pérdida

Para cada trayectoria τ , primero muestreamos el ruido $\epsilon \sim \mathcal{N}(0, I)$ y un paso de tiempo (timestep) $k \sim \mathcal{U}\{1, \dots, K\}$. La función de pérdida que se va a utilizar es la siguiente:

$$\mathcal{L}(\theta, \phi) := \mathbb{E}_{k, \tau \in \mathcal{D}, \beta \sim \text{Bern}(p)} [\|\epsilon - \epsilon_\theta(x_k(\tau), (1 - \beta)y(\tau) + \beta\emptyset, k)\|^2] + \mathbb{E}_{(s, a, s')} [\|a - f_\phi(s, s')\|^2]$$

Arquitectura

Parametrizamos ϵ_θ con la arquitectura U-Net que consiste en una red neuronal diseñada con bloques convolucionales residuales. Su arquitectura consiste en dos partes principales: un “codificador” y un “decodificador”. En nuestro problema, trataremos la secuencia de estados $x_k(\tau)$ como si fuesen imágenes donde la altura representa la dimensión de un único estado y la anchura la longitud de la trayectoria, así podemos trabajar perfectamente con dicha arquitectura.

Bibliografía

- [ADG⁺23] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is Conditional Generative Modeling all you need for Decision-Making? *arXiv*, 2023.
- [AN04] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 2004.
- [BDM17] Marc G. Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 06–11 Aug 2017.
- [BSA83] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.
- [Cou] Coursera. Reinforcement Learning Specialization. <https://www.coursera.org/specializations/reinforcement-learning>.
- [Die00] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.
- [Fos19] David Foster. *Generative Deep Learning*. O’Reilly Media, Inc., 2019.
- [Goo16] Ian Goodfellow. *Deep learning*. The MIT Press, Cambridge, Massachusetts, 2016.
- [Goo17] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2017.
- [GPAM⁺14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.
- [HE16] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [HZAL18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, 2018.
- [KL02] Sham Kakade and John Langford. Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.
- [Lap18] Maxim Lapan. *Deep Reinforcement Learning Hands-On*. Packt Publishing, Birmingham, UK, 2018.

- [LB19] J. Langr and V. Bok. *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning, 2019.
- [LHP⁺19] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv*, 2019.
- [Mat] MathWorks. Reinforcement Learning - MATLAB .
- [MKS⁺13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv*, 2013.
- [MKS⁺15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [Mor20] Miguel Morales. *Grokking Deep Reinforcement Learning*. Co., Manning Publications, 2020.
- [MPBM⁺16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.
- [Nie19] Frank Nielsen. On the Jensen–Shannon Symmetrization of Distances Relying on Abstract Means. *Entropy*, 21:485, 05 2019.
- [PC08] Fernando Perez-Cruz. Kullback-Leibler divergence estimation of continuous distributions. In *2008 IEEE International Symposium on Information Theory*, pages 1666–1670, 2008.
- [SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. MIT Press, 2018.
- [Sew19] Mohit Sewak. *Deep Reinforcement Learning - Frontiers of Artificial Intelligence*. Springer, 2019.
- [Sina] Sinergym. Sinergym Documentación. https://ugr-sail.github.io/sinergym/compilation/main/pages/notebooks/basic_example.html.
- [Sinb] Sinergym. Sinergym Github. <https://github.com/ugr-sail/sinergym>.
- [SLA⁺15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.
- [SWD⁺17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv*, 2017.

-
- [TTK⁺23] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.
- [Wan20] Yang Wang. A Mathematical Introduction to Generative Adversarial Nets (GAN). *arXiv*, 2020.
- [WD92] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.
- [ZB20] Alexander Zai and Brandon Brown. *Deep Reinforcement Learning IN ACTION*. MANNING SHELTER ISLAND, 2020.