UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA UDELAR

Planificación de vuelos para flotillas de vehículos autónomos aéreos usando redes neuronales generativas antagónicas

Felipe Almeida
Renzo Beux

Proyecto de grado presentado a la Facultad de Ingeniería de la Universidad de la República en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero en Computación

Tutores

Sergio Nesmachnow
Jamal Toutouh

Tribunal

Libertad Tansini
Laura González
Pedro Moreno

Montevideo, Uruguay
Noviembre de 2024

Resumen

La utilización de Unmanned Aerial Vehicles (UAV), como mecanismos de vigilancia, seguridad y otros fines, se ha vuelto cada vez más común. Para mejorar su eficiencia, se busca encontrar estrategias de recorrido que permitan visitar puntos de interés (POI), evitar obstáculos y mantenerse en vuelo con suficiente carga en las baterías para regresar a la base y recargarse. Además, se busca que los UAV sean capaces de minimizar el tiempo entre visitas a cada POI y aprovechar el tiempo restante para explorar la mayor cantidad posible de terreno.

En la actualidad, se utilizan algoritmos greedy para generar trayectorias de UAV en un espacio físico predefinido. Este enfoque implica dividir el espacio en cuadrantes, por los cuales los UAV se desplazan siguiendo los centroides de dichos cuadrantes. Sin embargo, cuando los cuadrantes representan áreas extensas, los UAV pueden no cumplir adecuadamente con su servicio en los POI ubicados en dichos cuadrantes. Para solucionar esto, es necesario mejorar la resolución de las trayectorias, es decir, reducir el tamaño de los cuadrantes. Sin embargo, esto genera un mayor número de cuadrantes y el algoritmo greedy comienza a aumentar el tiempo de cálculo y los requerimientos de memoria para generar las trayectorias.

El proyecto se basa en la idea de aumentar la resolución de las trayectorias de UAVs en un área preseleccionada. Se investiga el uso de Redes Generativas Antagónicas (GAN), una tecnología que ha experimentado un crecimiento significativo en los últimos años, para lograr el aumento de resolución. La propuesta principal consiste en entrenar una GAN con un conjunto de datos de trayectorias generadas mediante algoritmos greedy. El generador de la red sería capaz de generar trayectorias de mayor resolución, que luego se escalarían para que coincidan con el tamaño del conjunto de datos existente y así ser evaluadas por el componente discriminador. De esta manera, el generador puede producir una trayectoria de alta resolución que permite una mayor precisión en los movimientos del UAV.

El desarrollo del algoritmo greedy se llevará a cabo utilizando Python 3 (Python Software Foundation, 1991), y se aprovecharán las bibliotecas Tensorflow, Keras (Abadi y cols., 2015) y Pytorch (Paszke y cols., 2019), para el modelado de la GAN. A lo largo del proyecto se han realizado experimentos con diversas configuraciones, modelos y secuencias, buscando no solo el funcionamiento, sino también la optimización del proceso. Esta tarea ha requerido un

arduo trabajo que ha culminado en una GAN cuya salida precisa de una serie de scripts para post-procesar y así lograr un resultado aceptable.

Palabras clave: GAN, UAV, Trayectorias, Generación de trayectorias, Proyectos de Grado, Computación

Índice general

Capítulo 1

Introducción

El desarrollo y la adopción de UAVs abre un horizonte de posibilidades y desafíos en múltiples campos, particularmente en la vigilancia de áreas extensas. Los UAVs se consideran herramientas eficaces para la vigilancia debido a su capacidad de desplazamiento aéreo, rapidez y eficiencia(Cuerno y cols., 2016). Sin embargo, la generación eficiente de trayectorias emerge como un problema fundamental para maximizar su rendimiento y optimizar su uso.

Este proyecto aborda el desafío de mejorar la resolución de estas trayectorias, con especial énfasis en la aplicación de UAVs para la vigilancia. Se busca ampliar la efectividad de los UAVs al cubrir adecuadamente los POIs, evadir obstáculos y gestionar eficientemente la carga de batería de los vehículos. La mejora en la resolución de la trayectoria es crucial no sólo para optimizar el rendimiento, sino también para incrementar la seguridad y efectividad del patrullaje.

El primer enfoque utilizado para resolver este problema se basa en algoritmos "greedy". Se crean distintas heurísticas para experimentar distintos resultados. Al algoritmo greedy se le añade una componente de aleatoriedad para permitirle al UAV desplazarse por el mapa recorriéndolo, también se lo conoce como "wandering" (palabra del inglés que significa "deambular" o "vagar"). Estos algoritmos ofrecen una simplicidad y eficiencia que permite generar trayectorias iniciales de baja resolución. Sin embargo, este proyecto explora el potencial de las GANs en la generación de trayectorias de alta resolución para UAVs, con el objetivo de mejorar la vigilancia aérea. Se toma como punto de partida un dataset generado a partir del algoritmo greedy, el cual es utilizado para el entrenamiento de la GAN. Se busca tener un dataset con trayectorias variadas y con varias muestras para el mejor entrenamiento de la red.

La metodología de investigación se diseña considerando diversos factores que pueden afectar la generación y optimización de trayectorias. Estos incluyen características de la batería de los UAVs, la ubicación y frecuencia de visita de los POIs, la ubicación y forma de los obstáculos en el área de vigilancia, y la posible interacción entre varios UAVs en la misma zona.

Durante la evaluación, se establecen métricas de calidad para comparar diferentes trayectorias. Estas métricas consideran aspectos como la cobertura del

área de vigilancia, la frecuencia de visita a los POIs y el uso correcto de la batería de los UAVs.

Para la arquitectura de la GAN se analizan varios acercamientos, entre ellos: W-GAN, LSTM GAN, GAN convolucionales, entre otras. Se termina utilizando una arquitectura de GAN convolucional que genera imágenes de las trayectorias. Para ello se convierte el dataset generado por el algoritmo greedy a imágenes de trayectorias y se entrena la GAN con dicho dataset. La salida de lo generado por esta GAN carece de la dimensión del tiempo ya que al ser una imagen de la trayectoria y no la secuencia de acciones o puntos no es explicita en el recorrido realizado. Para esto se utiliza una adaptación del algoritmo greedy que genera el dataset de entrada, para generar a partir de la salida de la GAN una trayectoria representada a partir de una secuencia de acciones y de esta forma obtener efectivamente la dimensión del tiempo.

Algo a destacar es la necesidad de un modulo de baja de dimensiones (en inglés Downscaler) el cual permite el correcto funcionamiento de la GAN, ya que esta necesita que el generador y el discriminador utilicen las mismas dimensiones para operar correctamente. Es por esto que se desarrolla este módulo el cual es implementado con Redes Neuronales para mantener la información de los gradientes durante el entrenamiento.

Capítulo 2

Marco Teórico

Este capítulo presenta una introducción a las temáticas de el uso de UAVs, heurísticas y algoritmos greedy, redes neuronales y GANs.

2.1. Vehículos Aéreos No Tripulados

En esta sección se realiza una breve introducción a los UAVs, detallando sus usos, clasificaciones y aplicaciones.

2.1.1. Innovación y usos actuales

Los UAVs, también denominados comúnmente como drones, representan una innovación tecnológica de creciente relevancia en la contemporaneidad. Su evolución vertiginosa ha propiciado la aparición de diversas aplicaciones en el entramado social y económico. Los UAVs, concebidos como aeronaves capaces de navegar el espacio aéreo prescindiendo de la presencia de un piloto humano, son dirigidos por medio de control remoto y también a través de sistemas automatizados de gestión (Acero y cols., 2023).

Este avance tecnológico ha abierto un amplio abanico de posibilidades en múltiples campos de acción. Los UAVs se han convertido en aliados estratégicos en tareas de vigilancia y seguridad, en ámbitos civiles y militares. Su capacidad para operar en entornos de difícil acceso o riesgo, así como su versatilidad para llevar a cabo misiones de reconocimiento, los posiciona como herramientas fundamentales en la gestión de crisis y emergencias.

Los UAVs han incursionado con éxito en sectores como la agricultura de precisión, la cartografía y la topografía, revolucionando los métodos tradicionales de trabajo. Su capacidad para captar imágenes aéreas de alta resolución y recopilar datos en tiempo real ha optimizado procesos y reducido costos en actividades como el monitoreo de cultivos, la evaluación de daños ambientales y la planificación urbana.

2.1.2. Tipos de UAVs

Existen múltiples tipos UAVs, entre ellos se destacan los comercialmente más usados que son aquellos de tipo helicóptero que despegan de forma vertical con 4 o más motores, lo que les permite a estos UAVs mantenerse en el aire en una posición estable y realizar maniobras de vuelo en todas las direcciones (adelante, atrás, a los costados y combinaciones). Estas maniobras se logra mediante la variación individualizada de la velocidad de cada motor. En la misma categoría se encuentran los UAVs helicópteros unieje, que tienen un funcionamiento y comportamiento similar a los helicópteros tradicionales, con un solo rotor principal y un motor de cola para compensar la guiñada (movimiento en el eje vertical). Por otro lado, también existen UAVs con hélice frontal o turbinas, conocidos como UAVs de ala fija. Estos drones tienen un funcionamiento similar al de un avión, ya que requieren una pista para el despegue y aterrizaje (Cuerno y cols., 2016).

Cada tipo de UAV tiene sus propios usos y aplicaciones, dependiendo de su diseño y capacidades. Dentro del marco de este proyecto se trabaja con los UAVs de tipo helicóptero con múltiples motores, ya que son los más usados para vigilancia comercial/personal. Estos UAVs se caracterizan por su estabilidad y mayor capacidad de carga, lo que los hace ideales para este tipo de tareas. Además, pueden ser utilizados para una variedad de aplicaciones, como inspeccionar infraestructuras, monitorear cultivos o realizar búsquedas y rescates.

2.1.3. Aplicaciones de los UAVs

Los UAVs se utilizan en una amplia variedad de campos, desde aplicaciones militares y de seguridad, hasta aplicaciones civiles y comerciales. Entre las aplicaciones posibles se incluye: (Pombo, 2023)

- Entrega de paquetes: existen varias iniciativas que utilizan UAVs para la entrega de paquetes en varios contextos. Entre los más importantes y destacables son la entrega de suministros médicos en lugares inaccesibles o difícilmente accesibles por medios terrestres.
- Monitoreo agrícola: las imágenes y datos exactos de los cultivos son recolectados por los UAVs, facilitando a los productores agrícolas la optimización en la utilización de recursos y el seguimiento minucioso del desempeño de los cultivos.
- Mapeo y fotografía: los UAVs son capaces de producir mapas tridimensionales y representaciones topográficas de gran exactitud, elementos críticos para la organización urbana, la administración de recursos naturales y la detección de zonas de peligro.
- Inspección de infraestructuras: los UAVs tienen la capacidad de alcanzar zonas de acceso complicado para llevar a cabo evaluaciones visuales, contribuyendo a la identificación de desperfectos o cuestiones de mantenimiento en estructuras como puentes, torres y edificios de gran altura.

- Vigilancia y seguridad: los UAVs, dotados de cámaras y tecnología de reconocimiento, tienen la capacidad de supervisar vastas regiones, como límites territoriales o reservas naturales, y proporcionar asistencia en actividades de búsqueda y salvamento.

- Cinematografía: muchas películas incluyen tomas aéreas realizadas con UAVs equipado con cámaras de alta calidad, especializado en cinematografía. Estas tomas antes se realizaban desde helicópteros.

La versatilidad y el potencial de los UAVs los convierten en una herramienta valiosa en muchos sectores.

2.1.4. Desafíos en la operación de UAVs

A pesar de su utilidad y versatilidad, la operación de UAVs presenta varios desafíos. Uno de los principales desafíos es la planificación de trayectorias, que es esencial para garantizar que los UAVs puedan realizar sus misiones de manera segura y eficiente. La planificación de trayectorias implica determinar una trayectoria óptima para el UAV, evitando obstáculos y teniendo en cuenta las limitaciones del vehículo y del entorno. En el contexto de este proyecto, se investiga el uso de GANs para mejorar la planificación de trayectorias de los UAVs.

Un desafío particular que se presenta a la hora de trabajar con UAVs es el manejo de la batería. A pesar de que al día de hoy algunos UAVs comerciales pueden mantener un tiempo de vuelo de hasta 30 minutos, este tiempo es muy difícil de predecir, debido a que se ve muy afectado por las condiciones en las que se encuentra (viento, temperatura, peso del UAV) (Mansouri y cols., 2017). Debido a la dificultad para predecir la duración de la batería, en este proyecto se trabaja siempre manteniendo una correlación entre el tiempo de carga (TC) de un UAV y su tiempo de vuelo (TV), de $TV = 1{,}5 \times TC$.

2.2. Algoritmos Greedy y heurísticas

En el contexto de soluciones a problemas, la búsqueda de soluciones eficientes y precisas a problemas complejos es una tarea fundamental. En este sentido, los algoritmos greedy y las heurísticas se presentan como herramientas valiosas para abordar este desafío, ofreciendo un enfoque práctico y pragmático que complementa los métodos de búsqueda exhaustiva tradicionales. Los métodos de búsqueda tradicionales pueden requerir cantidades inmensas de computo y tiempo para hallar la solución, es aquí donde las heurísticas y algoritmos greedy juegan un papel importante

En esta sección se definen los conceptos de heurísticas y algoritmos greedy. Se establece la relación entre algoritmo greedy y heurística y se provee de un ejemplo de un algoritmo greedy muy conocido en el ámbito de búsqueda de trayectorias.

2.2.1. Heurísticas

Las heurísticas son reglas o métodos que ayudan a resolver problemas, en especial cuando una solución óptima es difícil de obtener o lleva mucho tiempo de procesamiento. En el contexto de la planificación de trayectorias, las heurísticas permiten tomar decisiones que se acercan al objetivo de manera más eficiente, aunque la solución pueda no ser óptima.

Las heurísticas emplean reglas prácticas para buscar el camino más corto hacia el objetivo. Este tipo de algoritmos se utilizan ampliamente en la planificación de trayectorias, particularmente en entornos discretos.

A pesar de la eficacia de las heurísticas, pueden tener limitaciones, especialmente en entornos complejos, dinámicos o de gran escala. En ciertos contextos, el uso de una heurística para la búsqueda de una solución óptima no es práctica ni eficiente. En su lugar, se emplean estrategias que buscan soluciones aproximadas, adaptativas, o que descomponen el problema, utilizando técnicas como el aprendizaje automático para generar soluciones de forma efectiva.

2.2.2. Algoritmos Greedy

Un algoritmo greedy consiste en un procedimiento iterativo, que toma decisiones contemplando únicamente parte del contexto en el que se encuentra. En el contexto de planificación de trayectorias, esto implica no contemplar la totalidad del ambiente, por ejemplo ignorando a otros navegadores del entorno, o algún obstáculo presente en el ambiente.

Los algoritmos greedy son excelentes para resolver problemas con bajo costo computacional. Son simples de entender y, en ciertos tipos de contextos, encuentran soluciones que son óptimas. Debido a la naturaleza local de la toma de decisiones en los algoritmos greedy, estos algoritmos a menudo llegan a soluciones subóptimas o, en algunos casos, no garantizan una solución.

2.2.3. Relación entre Algoritmos Greedy y Heurísticas

Los algoritmos greedy son una implementación de una heurística específica. En el contexto de la planificación de trayectorias, la heurística puede consistir en avanzar siempre en la dirección que minimiza la distancia al objetivo, y el algoritmo greedy constituye la implementación de esta heurística.

En resumen, las heurísticas y los algoritmos greedy son estrategias prácticas para resolver problemas de búsqueda y optimización que para ciertos escenarios son demasiado costosos para resolver exactamente. La elección de la heurística o algoritmo a utilizar depende de la naturaleza del problema y del contexto en el que se utiliza.

2.2.4. Algoritmo A*

El algoritmo A* es un algoritmo greedy de la categoría “buscadores de trayectorias” o mejor conocido en inglés como Pathfinding Algorithms. Es el algoritmo más popular en el área debido a su flexibilidad y poder de adaptación

para distintos contextos. Es una combinación y adaptación de dos algoritmos muy conocidos: Dijkstra y Best-First-Search(Patel, 2024).

En cada iteración A* explora el nodo que minimiza *f(n) = g(n) + h(n)*, donde:

- $g(n)$ es la función que representa el costo de un camino realizado desde el punto de inicio hasta cualquier nodo n
- *h(n)* es la heurística para calcular el costo estimado desde un vértice n al vértice de destino. Es estimado porque hasta no recorrer la trayectoria hasta el final no se sabe que obstáculos pueden impedir la trayectoria. Esta heurística puede ser la función de distancia Euclidiana o Manhattan al vertice destino.

En términos generales, el algoritmo A* permite obtener una trayectoria hacia el objetivo que se aproxima a la óptima, siempre que la heurística de distancia definida sea coherente con la verdadera distancia al objetivo. Si la heurística se desvía significativamente del costo real, aumenta la probabilidad de que la trayectoria resultante no sea óptima.

La figura 2.1 muestra una instancia donde, debido a los valores que toma la heurística $h(n)$ para ciertos nodos, el camino encontrado no es el óptimo. El algoritmo A* recorre la arista inferior debido a que la formula $f(n) = g(n)+h(n)$ resulta en cinco utilizando la arista inferior desde el punto S, mientras que por el camino superior resulta en siete. En cambio, si se recorre todo el camino superior el costo total es de siete, mientras que recorriendo el camino inferior por completo el costo resulta en ocho. Esto se debe a que la heurística de costo no cuenta con la propiedad de consistencia, la cual es la siguiente: $\forall n, n' \in V$, donde n y n' son nodos adyacentes $h(n) \leq c(n, n') + h(n')$ donde $c(n, n')$ es el costo real de ir de n a n'. Fue demostrado por Rina Dechter y Judea Pearl (Dechter y Pearl, 1985) que el algoritmo de A* con una heurística consistente es óptimo.

Un escenario usual de recorrida de trayectoria consiste en una grilla de cuadrantes, donde están permitidos los movimientos hacia cuadrantes adyacentes, y se cuenta con algún obstáculo dentro de la grilla que impide el movimiento. A menos que haya alguna restricción dentro de los cuadrantes, se suele tener que el costo de movimiento entre adyacentes es 1, y luego se puede usar como estimado de distancia a destino, la distancia real entre el punto actual y el destino. Si los movimientos permitidos son en las cuatro direcciones cardinales, entonces una buena estimación de la distancia hasta el objetivo es la distancia de Manhattan. Si además se permiten movimientos en las 4 diagonales, una mejor estimación del costo hasta el objetivo es la distancia Euclidiana.

2.3. Redes neuronales

En esta sección se realiza una introducción a las redes neuronales, detallando su composición, su historia en los ultimo años, y se describen distintos tipos de redes neuronales

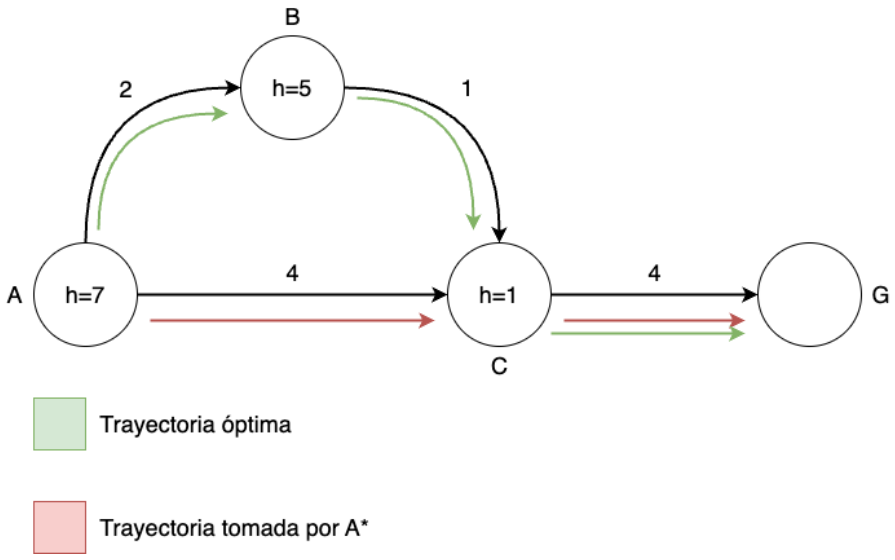


Figura 2.1: Trayectoria subóptima A*

2.3.1. Composición y ecosistema

Las redes neuronales son modelos computacionales inspirados por el funcionamiento del cerebro humano. Consisten en múltiples capas de neuronas, que se encuentran interconectadas de una capa a la siguiente. Cada neurona cuenta con un vector de pesos que utilizan para combinar con la salida de las neurona de la capa previa y calcular su salida.

Una estructura común para las redes neuronales es la del perceptrón multicapa. En esta estructura, cada neurona i realiza una combinación lineal de todas las salidas de la capa previa, vector x_i, con los pesos de la neurona, vector w_i, a lo que se le suma un término independiente l_i. El resultado de esta operación se pasa a través de una función de activación $\sigma(x)$, obteniendo así el resultado de la neurona $f_i(x_i) = \sigma(w_i x_i + l_i)$.

Debido a la accesibilidad del hardware necesario para llevar a cabo el entrenamiento de las redes neuronales, hoy en día existe un extenso y robusto ecosistema de bibliotecas que facilitan el desarrollo y entrenamiento de redes neuronales artificiales. Este rico conjunto de herramientas y recursos proporciona a los desarrolladores una amplia gama de opciones para implementar y experimentar con diversos tipos de arquitecturas y enfoques de aprendizaje profundo. Esto ha llevado a la estandarización de arquitecturas de redes neuronales mediante el uso de bibliotecas que ya cuentan con la implementación de múltiples partes de las redes. Ejemplos de estas bibliotecas son Keras y Pytorch; estas bibliotecas cuentan con módulos para construir redes neuronales.

2.3.2. Historia y aplicaciones

El uso de redes neuronales se ha expandido a numerosos ámbitos, entre ellos el de los videojuegos. El rápido avance en la tecnología gráfica de los juegos ha impuesto exigencias cada vez mayores en términos de rendimiento y calidad visual. Sin embargo, las limitaciones físicas de los dispositivos muchas veces impiden cumplir con estos requisitos. Para abordar este desafío, empresas como Nvidia han desarrollado soluciones innovadoras como Deep Learning Super Sampling (DLSS). Esta tecnología emplea redes neuronales para realizar un reescalado inteligente de las texturas, lo que permite mejorar el rendimiento en tiempo real sin comprometer la calidad visual (Watson, 2020).

Una de las aplicaciones más importantes en la actualidad es el desarrollo de generadores de contenido como puede ser ChatGPT, Gemini o DALL-E, entre otros. Estos sistemas permiten la generación de código, textos, imágenes e incluso vídeos. Aplicaciones que no solo están enfocadas en el uso empresarial sino que en el uso personal y diario. Sin embargo la popularización de estas aplicaciones están haciendo evidente el poder computacional todavía necesario para llevar a cabo las tareas que estos sistemas ofrecen. Además, es importante señalar las preocupaciones ambientales que acompañan este desarrollo, así como la necesidad de continuar investigando en este campo (Gordon, 2024)

La alta demanda por el uso de redes neuronales lleva al desarrollo de hardware especializado en su entrenamiento y uso. Un claro ejemplo de mejoras de hardware para redes neuronales son la introducción tensor cores por parte de Nvidia, núcleos de GPUs especializados en el manejo de tensores, que prometen aceleraciones de hasta cuatro veces la original. Otro elemento de hardware especializado para el entrenamiento de redes neuronales son las Tensor Processing Units (TPUs) diseñadas por Google en 2015, que sacrifican precisión en operaciones aritméticas para aumentar su velocidad. Este hardware comienza a ser ofrecido como un servicio en la nube por organizaciones como Google, Amazon y Microsoft, permitiendo que más personas sean capaces de entrenar redes neuronales. El análisis y desarrollo de hardware relacionado a redes neuronales subraya su importancia el mundo de la tecnología.

2.3.3. Aprendizaje Profundo

El aprendizaje profundo es un subcampo del aprendizaje automático que se centra en el uso de redes neuronales con muchas capas, también conocidas como “redes neuronales profundas”. Estas redes neuronales profundas pueden modelar relaciones complejas entre las entradas y las salidas, y son especialmente buenas para aprender a partir de grandes cantidades de datos (Goodfellow y cols., 2016).

En la última década, las técnicas de aprendizaje profundo han revolucionado muchos campos, incluyendo la visión por computadora, el procesamiento del lenguaje natural, y la robótica. Estos avances se deben en gran parte a la disponibilidad de grandes cantidades de datos de entrenamiento y al aumento en la potencia de cálculo, que ha hecho posible entrenar redes neuronales profundas (Goodfellow y cols., 2016).

2.3.4. Redes neuronales recurrentes

Dentro de muchos contextos distintos de entrenamiento de redes neuronales se pueden hallar datos de entrenamiento que son temporalmente dependientes, refiriéndose a que la ocurrencia de cierto patrón en los datos tiene distinto significado dependiendo de donde se ubica el mismo dentro del conjunto de datos.

Para abordar este problema se introducen las redes neuronales recurrentes (RNN). Las mismas son redes neuronales a las cuales se le suma un estado escondido. Este estado escondido es considerado como parte de la entrada de la red neuronal en cada paso de su entrenamiento y evaluación, pero se ve definido por el historial del modelo de aprendizaje. Esto significa que una red neuronal puede retornar para exactamente los mismos parámetros de entrada, dos salidas distintas, ya que depende del historial que haya tenido.

De forma similar a las redes con un gran número de capas, o profundas, al intentar detectar dependencias de largo alcance estas redes sufren de un problema conocido como "gradiente desvanecente", donde el gradiente que fluye a través de la red se vuelve demasiado pequeño, al punto de que afecta muy poco a las primeras capas de la red. Para atacar este problema se desarrollaron arquitecturas de redes neuronales recurrentes capaces de olvidar y recordar información de forma más selectiva. Algunos ejemplos de estas arquitecturas son las capas Long Short-Term Memory (LSTM) (Hochreiter y Schmidhuber, 1997) o las Gated Recurrent Unit (GRU) (Bahdanau y cols., 2016)

2.4. Redes Generativas Antagónicas

Las GANs fueron introducidas por Ian Goodfellow y sus colegas en 2014, representando un importante avance en el campo de la inteligencia artificial. En términos generales, las GANs son modelos de aprendizaje automático con la capacidad de generar nuevos datos que guardan una gran similitud con los datos con los que fueron entrenadas (Goodfellow y cols., 2014) , lo que hace a las GANs ideales para una amplia gama de aplicaciones, desde la creación de arte hasta la generación de datos sintéticos para la formación de otros modelos de aprendizaje automático.

En esta sección se introduce las componentes y arquitectura de las GANs así como distintos tipos de GANs comúnmente usadas y sus características.

2.4.1. Componentes de las GANs

Una GAN está compuesta por dos elementos fundamentales:

- **El discriminador**, modelo de clasificación encargado de clasificar datos o instancias del sujeto de estudio como reales o generadas. El discriminador se entrena con un set de datos reales que utiliza para comparar con las próximas instancias de datos que se le presenten, para clasificarlas como

reales o generadas. El papel del discriminador en este proyecto es identificar si las trayectorias de vuelo generadas se asemejan al conjunto de trayectorias de vuelo previamente validadas.

- **El generador**, modelo responsable de producir nuevos datos, a partir de un vector de ruido aleatorio, que se asemejen lo más posible a los datos de entrenamiento. El objetivo de este modelo será producir instancias de datos clasificadas como reales por el discriminador. En el caso de esta proyecto, el objetivo del generador es la producción de trayectorias de vuelo plausibles para los UAVs.

En un primer momento la GAN genera datos aleatorios. Para que lo generado por la GAN se asemeje al objetivo, debe ser sometida a un entrenamiento. En el proceso de entrenamiento de una GAN, el generador y el discriminador interactúan de manera iterativa. El generador crea datos sintéticos que se entremezclan con datos reales, y el discriminador tiene la tarea de clasificarlos como reales o falsos. A partir de estas clasificaciones, el generador ajusta su modelo con el objetivo de producir resultados que sean cada vez más efectivos en engañar al discriminador. Paralelamente, el discriminador se entrena continuamente, mejorando su capacidad para distinguir entre los datos reales y los generados, basado en su rendimiento en estas clasificaciones. Este proceso puede entenderse como un juego de suma cero (MasterClass, 2023), en el que las mejoras del generador en producir datos convincentes se traducen en un desafío creciente para el discriminador, y viceversa, generando un equilibrio dinámico en el que los avances de uno suponen pérdidas para el otro. La figura 2.2 muestra el flujo de entrenamiento de una GAN.

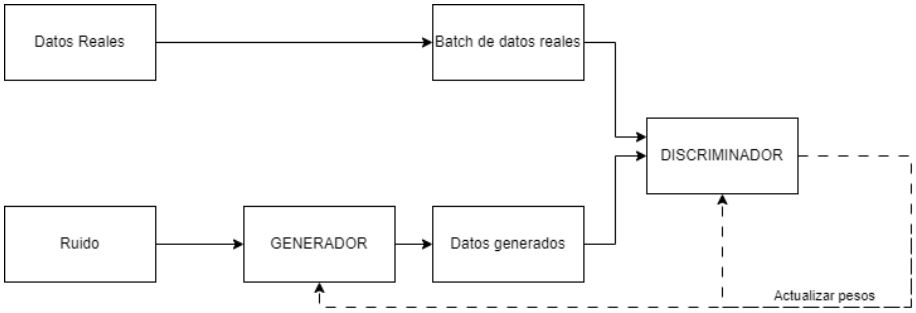


Figura 2.2: Flujo de entrenamiento de una GAN

2.4.2. Entrenamiento de una GAN

En esta sección se presenta el ciclo de entrenamiento de una GAN y sus componentes relevantes.

Entrenamiento

El generador y el discriminador de una GAN son redes neuronales capaces de operar de forma independiente, que utilizadas en conjunto conforman la GAN. En el caso del generador, lo que recibe es un vector de ruido aleatorio que una vez transformados se intentan asemejar al sujeto de estudio, en el contexto de este proyecto, trayectorias de UAVs. Mientras tanto el discriminador recibe datos, ya sean reales, o generados por el generador, y para cada uno retorna una clasificación entre real o generada, con una cierta confianza, un valor entre cero y uno donde cero indica generado y uno real.

El entrenamiento de una GAN implica mejorar el generador y el discriminador de manera iterativa, de tal forma que el generador sea capaz de producir datos cada vez más convincentes y el discriminador se vuelva más preciso en distinguir los datos reales de los generados. Se alcanza la convergencia cuando se logra que el discriminador acierte al 50 % de sus clasificaciones, implicando que no es posible para el discriminador distinguir entre lo generado de lo real, dando una posibilidad de acertar de 50 %.

Algoritmo 1 Entrenamiento de GAN

```
1: Entrada: variables de ambiente, generador (Gen), discriminador (Disc)
2: Salida: generador y discriminador entrenados
3: procedure ENTRENAR_EPOCA
4:     Inicializar losses acumulativas en 0 y evaluación promedio en 0
5:     for batch in batches do
6:         for k veces do
7:             Generar datos falsos con generador
8:             Calcular pérdida del discriminador con datos falsos y reales
9:         end for
10:        Generar datos falsos con generador
11:        Calcular pérdida del generador con el discriminador y datos falsos
12:        Actualizar pérdidas acumulativas del generador y discriminador
13:    end for
14:    Calcular pérdidas promedio y evaluación promedio
15:    return Pérdida promedio de generador, pérdida promedio de discriminador y evaluación promedio
16: end procedure
17: procedure ENTRENAMIENTO
18:    Inicializar Gen, discriminator (Disc)
19:    for época in épocas do
20:        {Llamada a ENTRENAR_EPOCA}
21:        Registrar pérdida de discriminador, pérdida de generador y evaluación promedio
22:    end for
23: end procedure
```

En el algoritmo 1 se muestra el proceso general de entrenamiento de una GAN, donde se usa la variable K para repetir el entrenamiento de del discriminador. Esto es un hiperparámetro muy usado en el entrenamiento de las GANs. Permite entrenar el discriminador k veces más que el generador. En algunos casos es necesario ya que al estar el discriminador menos entrenado que el generador, este último no otorga información valiosa.

Función de pérdida, optimizadores y Learning Rate

Las GANs buscan minimizar dos funciones de pérdida: la del discriminador y la del generador. La función de pérdida es una métrica numérica que cuantifica la discrepancia entre la salida de la red y el valor objetivo. Para el discriminador, esta función evalúa qué tan eficazmente puede distinguir entre datos reales y datos generados por el generador. En contraste, para el generador, la función de pérdida indica qué tan exitosamente engaña al discriminador produciendo datos que parecen reales. Estas funciones de pérdida pueden ser definidas a medida para incluir otros aspectos como por ejemplo usando funciones de evaluación sobre la salida del generador para motivar al generador a generar salidas que satisfagan a ese evaluador.

Una componente importante de una GAN son los optimizadores que desempeñan un papel crucial en el entrenamiento de las GANs al ajustar los parámetros del modelo para minimizar las funciones de pérdida . Esto mejora la calidad y precisión del modelo, tanto el generador como el discriminador tiene su propio optimizador que ajusta los parámetros de la red neuronal especifica. Algunos optimizadores ampliamente utilizados en la literatura son RSMProp, ADAM y SGD.

Las redes neuronales buscan minimizar su función de pérdida. La minimización se busca mediante la actualización de los pesos de las neuronas que componen la red acorde a la fórmula $w^{t+1} = w^t - \eta \cdot \nabla_w L$ donde w^t es el peso de una neurona en la iteración t de entrenamiento $\nabla_w L$ es el cálculo del gradiente de la función de pérdida, acorde según los pesos, y η es el hiperparámetro learning rate (LR). La fórmula muestra como la magnitud del LR es lo que dicta que tanto varían los pesos de los nodos en cada iteración. En resumen, el LR es lo que se encarga de dictar la velocidad con la cual aprenden las redes neuronales. Un LR muy bajo, lleva a que el entrenamiento de las redes sea extenso, mientras que uno muy alto lleva a oscilaciones en el entrenamiento, dado a que la velocidad no permite la convergencia al mínimo de la función de pérdida.

Un Learning Rate Scheduler (LRS) es un algoritmo que permite la variabilidad del Learning Rate (LR) de un modelo de aprendizaje automático a lo largo del entrenamiento. Si bien existen varios tipos de LRS algunos de los más comunes son:

- **LRS basado en tiempo:** estos LRS varían el LR conforme transcurren las épocas de entrenamiento. El ajuste puede ser lineal, logarítmico, exponencial, entre otros. Un ejemplo de esto es $lr = lr_0 \times e^{(-kt)}$, donde lr_0 es el Learning Rate inicial, k es un hiperparámetro y t es la época. En

este caso, el LR disminuye de forma exponencial a medida que pasan las épocas. Los LRS basados en tiempo son simples de entender.

- **LRS basados en el rendimiento:** estos LRS disminuyen o aumentan el LR basado en el rendimiento del modelo. Es común verlos implementados utilizando la pérdida del modelo o alguna función de evaluación del resultado del mismo. Los LRS basados en rendimiento ofrecen varias ventajas:
  - **Adaptación Dinámica:** estos LRS ajustan la tasa de aprendizaje de manera dinámica durante el entrenamiento. Esto significa que el LR puede disminuir si el rendimiento del modelo mejora y aumentar si el rendimiento empeora, lo que permite una adaptación continua a medida que el modelo converge.
  - **Optimización Eficiente:** al ajustar el LR en función del rendimiento del modelo, los LRS basados en rendimiento pueden ayudar a optimizar el proceso de entrenamiento de manera más eficiente. Por ejemplo, si el modelo muestra un buen rendimiento en las últimas etapas del entrenamiento, el LR puede reducirse para permitir una convergencia más precisa y evitar oscilaciones innecesarias.
- **LRS cíclicos:** estos schedulers ajustan el LR de forma cíclica, variándolo entre un máximo y un mínimo. La idea detrás de estos LRS es aprovechar varios beneficios:
  - **Escape de Mínimos Locales:** los schedulers tradicionales de learning rate pueden causar que el proceso de optimización quede atrapado en mínimos locales o mesetas. Al ajustar periódicamente el LR, los LRS cíclicos permiten al optimizador escapar de estos mínimos locales
  - **Regularización:** la variación cíclica del LR puede actuar como una forma de regularización. Al introducir fluctuaciones en el proceso de aprendizaje, los LRS cíclicos agregan ruido al proceso de optimización. Este ruido puede ayudar a prevenir el sobre ajuste al desalentar al modelo de converger hacia soluciones demasiado complejas que se ajusten demasiado bien a los datos de entrenamiento. En su lugar, alienta al modelo a generalizar mejor con respecto a los datos no vistos.

2.4.3. Arquitecturas de GANs

En esta sección se introducen distintas arquitecturas de GANs, su funcionamiento, y sus fortalezas.

Redes convolucionales

Una alternativa para la estructura del generador es una red neuronal convolucional (CNN por su sigla en inglés). Las mismas son de gran utilidad a la

hora de trabajar con imágenes. Una red neuronal se considera convolucional en cuanto alguna/s de sus capas lleven a cabo una operación de convolución con un cierto kernel (Sousa, 2019). En la figura 2.3 se muestra una convolución matricial a modo de ejemplo:

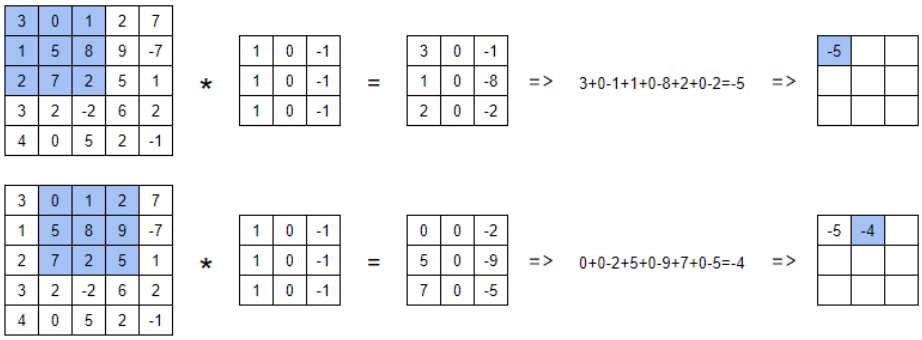


Figura 2.3: Ejemplo de convolución

Muchos kernels han resultado útiles para la detección de ciertas características de imágenes en particular, por lo que son reutilizados y en ocasiones forman parte de redes neuronales sin que las capas sean ajustadas. Ejemplos de este tipo de kernels incluyen el kernel para la detección de bordes 2.1, utilizado para resaltar los bordes en una imagen; el kernel para desenfocar imágenes 2.2, que se emplea para suavizar detalles y reducir ruido; y el kernel para resaltar detalles en una imagen 2.3, que incrementa la nitidez de los bordes y detalles.

$$\mathbf{K}_{\text{bordes}} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.1}$$

$$\mathbf{K}_{\text{desenfoque}} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.2}$$

$$\mathbf{K}_{\text{detallado}} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{2.3}$$

Wasserstein GAN

Entre las arquitecturas utilizadas para conseguir mejores resultados se encuentra la Wasserstein GAN, introducida en 2017 (Arjovsky y cols., 2017) como una variante de la arquitectura tradicional. En esta, se cambia el discriminador de una GAN tradicional por lo que se conoce como un crítico. El crítico cumple un rol muy similar al del discriminador en cuanto evalúa lo generado por el generador, pero se diferencian en cuanto el crítico retorna no una probabilidad como lo hace el discriminador, sino una estimación de la distancia

entre la distribución del dataset y la distribución de los datos generados por el generador. La función que trata de estimar el crítico se conoce como distancia de Wasserstein-1, o también Earth Mover's Distance (EMD). Esta función lleva a que el crítico tenga un codominio no acotado a diferencia del discriminador que siempre retorna valores de probabilidades ,entre cero y uno.

Las Wasserstein GAN apuntan a reducir la inestabilidad que suele presentarse en las redes generativas regulares y evitar problemas comunes como el colapso de modelo (Saatchi y Wilson, 2017). La desventaja que presentan las mismas es un elevado tiempo de entrenamiento, ya que en altas dimensiones, su estimación de la distancia de Wasserstein-1 se aleja más de su valor real.

RNN

En múltiples casos es de interés que las redes neuronales no establezcan una relación uno a uno entre su entrada y su salida. Un ejemplo claro de esto es el análisis de texto con fines predictivos. En estos contextos, el comienzo de una palabra no debería resultar siempre en el mismo final. Para esto se implementan las RNN que cuentan con memoria mediante la combinación de los datos de entrada con los datos de salida de la iteración previa. De esta forma la red cuenta con un contexto de lo procesado en previas iteraciones, y se puede expandir para que la red mantenga en memoria un mayor numero de iteraciones previas aumentando así lo que se conoce como el "contexto". La figura 2.4 muestra un diagrama de una neurona de red RNN, donde N representa la neurona, M representa la memoria de la red, I_t representa la entrada en el instante de tiempo t, y O_t representa la salida de la neurona en el instante de tiempo t.

Las RNN se utilizan para mejorar la capacidad de las redes neuronales para generar datos realistas y coherentes. Por ejemplo un generador puede ser modelado con una RNN para capturar dependencias temporales complejas entre los datos generados en diferentes etapas del proceso de creación de imágenes o secuencias. Esta capacidad permite a la GAN aprender y reproducir patrones de datos que mantienen coherencia y continuidad, de esta forma mejorando la calidad de las salidas generadas.

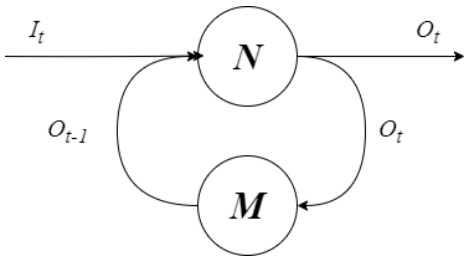


Figura 2.4: Ejemplo de neurona recurrente

LSTM-GAN

Es común querer estudiar series secuenciales de datos, como por ejemplo, estadísticas financieras a lo largo del tiempo. Esta necesidad lleva al diseño de una arquitectura de red neuronal diseñada específicamente para su análisis, conocida como LSTM-GAN. Fueron introducidas en 1997 por Josef Hochreiter et al., consisten en redes neuronales que cuentan con lo que se conoce como memoria interna en cada una de sus neuronas (Hochreiter y Schmidhuber, 1997). Las LSTM son una variación de las RNN. Para procesar los datos, cada neurona de una red neuronal LSTM hace uso de los datos provistos por la capa anterior, el resultado de la ultima operación realizada, y el estado actual de su memoria. En cada operación la neurona cuenta con mecanismos para actualizar selectivamente su memoria interna en función de los datos obtenidos y su estado actual, de esta forma reteniendo información relevante a lo largo del entrenamiento y la evaluación. La figura 2.5 muestra un esquema de una neurona de una red neuronal LSTM donde:

- C_t: representa la memoria selectiva de la neurona en la iteración t.
- H_t: representa los pesos de la neurona en la iteración t y también representa la sálida de la neurona.
- I_t: representa los datos a procesar por la neurona en la iteración t.
- $\times$ y $+$: representan operaciones de multiplicación y suma, respectivamente, elemento a elemento de las matrices.
- σ: representa funciones de activación sigmoide.
- *tanh*: representa una función de activación de tangente hiperbólica.

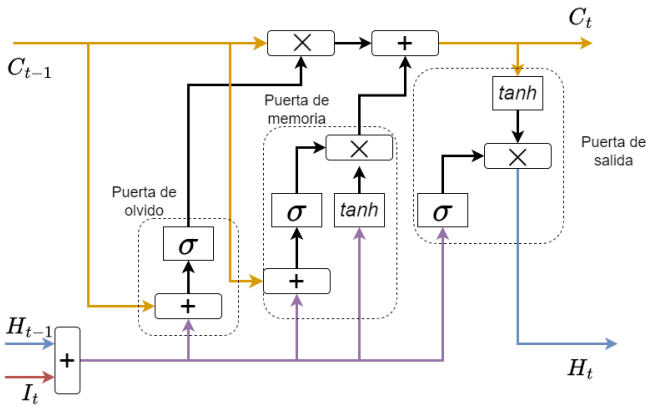


Figura 2.5: Ejemplo de neurona LSTM

Capítulo 3

Revisión de antecedentes

La inteligencia artificial ha encontrado altos niveles de uso en el campo de diseñado de trayectorias, debido al alto nivel de adaptabilidad al ambiente que es capaz de lograr. El uso de GANs en particular para la navegabilidad ya ha sido explorado por (Mohammadi y cols., 2018), pero la calificación del resultado final continua siendo altamente dependiente de feedback humano.

Se ha estudiado la planificación de trayectorias de UAVs mediante el uso de Reinforcement Learning (Puente-Castro y cols., 2022), más específicamente mediante el uso de Q-Learning. Dentro de las conclusiones presentadas por el investigación se mencionóel uso de una única GAN que englobe a todos los UAVs presentes en el contexto en contraposición a la alternativa que encontraron ser menos eficiente, una GAN por UAV. Los resultados de esta investigación respecto al uso de una Red Neuronal Artificial, por sus siglas en ingles ANN, global son de importancia para las decisiones a tomar en este proyecto.

En el artículo "Autonomous flight of unmanned aerial vehicles using evolutionary algorithms" (Gaudín y cols., 2019) se presentó un enfoque para la planificación de flotas de UAVs en donde se utilizaron algoritmos evolutivos para realizar esta planificación buscando maximizar simultáneamente tres funciones: el beneficio de tener visión de un objetivo, el de formar una red ad-hoc y el de explorar el área. Se generaron instancias de problemas sintéticos y se analizaron configuraciones de forma detallada tanto para el algoritmo evolutivo como para Mutation and Selection Only Evolutionary Strategy (MOSES), un algoritmo de optimización multiobjetivo. Los resultados reportados en el articulo muestran que el algoritmo propuesto basado en el algoritmo evolutivo fue 2.2 veces más preciso que MOSES y el algoritmo greedy. El algoritmo propuesto supera a MOSES y al greedy en cada ejecución al resolver dos de las tres instancias de problemas consideradas para el análisis experimental. Si bien el algoritmo evolutivo y MOSES tienen una mejor precisión en los resultados generados que el algoritmo greedy, el algoritmo greedy resuelve el problema en 0.1 segundos en comparación a los seis minutos que demoran los otros dos algoritmos. La velocidad del algoritmo greedy lo hace un buen candidato para soluciones en tiempo real. Este articulo demuestra la viabilidad de construcción

de trayectorias de UAVs mediante algoritmos greedy, y como pueden obtenerse mejores trayectorias mediante el uso de técnicas de aprendizaje automático.

El artículo "Distributed greedy approach for autonomous surveillance using unmanned aerial vehicles" (Behak y cols., 2020) continuó el trabajo presentado en "Autonomous flight of unmanned aerial vehicles using evolutionary algorithms". Este último concluyó que la utilización de algoritmos greedys es útil por su rapidez en la planificación en tiempo real de las trayectorias, debido a este hallazgo Behak et al. desarrollaron e implementaron un algoritmo greedy capaz de organizar distintos UAVs en pos de vigilar un área de forma autónoma. La solución propuesta consiste en que cada UAV tome su propia decisión en base a los datos que tiene disponible y los obtenidos a partir de la comunicación con los otros UAVs. Para lograr estas decisiones se propuso una máquina de estados en donde dependiendo de la situación el UAV toma distintas acciones. La experimentación fue llevada a cabo con 3 UAVs, en escenarios reales y escenarios sintéticos. Para evaluar la eficacia del algoritmo greedy los autores lo comparan con otros dos algoritmos, exploración por regiones y caminata aleatoria. Los resultados obtenidos muestran que el algoritmo greedy superó a los otros dos algoritmos tanto en la métrica de cobertura como en la de vigilancia.

Liu et al. propuso en el ámbito de la protección de la privacidad geográfica y generación de datos sintéticos a partir de datos reales con el fin de mantener una consistencia en las propiedades estadísticas obtenidas de los datos reales una arquitectura especifica para la generación de trayectorias a la cual nombró trajGAN (Liu y cols., 2018). Esta investigación es de vital importancia ya que en muchos casos se quieren utilizar datos reales para distintos estudios pero por motivos de privacidad esto se dificulta ya que muchas partes pueden obtener información sensible de varios actores. El uso de GANs con este fin representa un avance en la protección de la privacidad geográfica. En el marco del proyecto provee información pertinente para el desarrollo de la GAN propuesta, ya que aporta ideas de como poder abordar la arquitectura de la solución.

Una solución especifica al problema de la generación de datos para la ocultación de información real por motivos de privacidad fue propuesta por Rao et al. (Rao y cols., 2020). La solución utiliza una arquitectura de GAN con técnicas de Long Short-Term Memory (LSTM) para la generación de estas trayectorias, con el fin de ocultar información a aquellas entidades que utilicen estos datos con motivos de análisis. Los objetivos propuestos por Rao et al. fueron alcanzados generando trayectorias la cuales mantenían las cualidades estadísticas del dataset y al mismo tiempo eran artificiales, esto es, que no se podía asociar la trayectoria a un usuario especifico protegiendo así la privacidad de los usuarios que reportan información para su estudio.

La tabla 3.1 ofrece un resumen conciso de los trabajos afines, ordenados según su mención previa. Se proporciona una breve descripción destacando la contribución principal de cada trabajo en el contexto del proyecto de grado. Este relevamiento de estudios similares permite analizar los objetivos y desafíos abordados en otras investigaciones del área.

Trabajo	Comentarios
Mohammadi y cols. (2018)	Utiliza GANs con el fin de generar trayectorias desde un punto de inicio hasta un punto de finalización. Obtiene buenos resultados y enfatiza en que el mejor evaluador es el feedback humano.
Puente-Castro y cols. (2022)	Se utiliza técnicas de reinforcement learning para generar trayectorias. Logran resultados satisfactorios.
Gaudín y cols. (2019)	Utiliza algoritmos evolutivos para la planificación de trayectorias. Los resultados muestras soluciones precisas por parte del algoritmo evolutivo pero mayor rapidez en el algoritmo greedy
Behak y cols. (2020)	Continuación de Gaudín y cols. (2019) donde se desarrolla un algoritmo greedy, que basándose en una máquina de estados supera a exploración por regiones y caminata aleatoria.
Liu y cols. (2018)	Se emplean GANs para la generación de datos sintéticos que mantengan propiedades estadísticas presentes en los datos originales.
Rao y cols. (2020)	Plantea el uso de capas LSTM para la generación de trayectorias sintéticas. Es un avance sobre el trabajo de Liu et al.

Cuadro 3.1: Resumen de trabajos relacionados

Capítulo 4

Propuesta

En este capitulo se definen todos los elementos que constituyen una instancia del problema, el problema a resolver, el proceso mediante el cual se generan trayectorias y el método empleado para la mejora de resolución de las trayectorias.

4.1. Definiciones

En esta sección se definen los componentes con los que se trabaja en la solución de este proyecto.

Área de vigilancia. El área de vigilancia es el terreno a vigilar, donde los UAVs llevan a cabo su patrullaje. El área se divide en una cuadrícula, cuyas dimensiones se conocen como dim_x y dim_y. La ubicación de partida de los UAVs dentro del área se encuentra en la base, que se sitúa en los puntos (b_x, b_y) del área de vigilancia. Todos los UAVs comienzan sus trayectorias desde esta posición. Los UAVs se mueven entre cuadrantes adyacentes, posicionándose en el centroide de cada cuadrante.

UAV. La vigilancia del área es realizada por un conjunto de UAVs, representado por U. Cada UAV tiene una trayectoria definida. Cada UAV cuenta con una batería que le permite volar durante T_v instantes de tiempo. Cuando la batería llega a un nivel en el que requiere ser cargada, el UAV vuelve a la base. El tiempo de recarga es de T_r, durante el cual el UAV se considera estático en la base.

Trayectoria. Una trayectoria se compone de una secuencia de movimientos, donde cada movimiento es uno de los nueve posibles listados a continuación:

0. Mantenerse en el cuadrante actual $(+0, +0)$
1. Derecha $(+1, +0)$

2. Diagonal inferior derecha $(+1, -1)$
3. Abajo $(+0, -1)$
4. Diagonal inferior izquierda $(-1, -1)$
5. Izquierda $(-1, +0)$
6. Diagonal superior izquierda $(-1, +1)$
7. Arriba $(+0, +1)$
8. Diagonal superior derecha $(+1, +1)$

A cada UAV se le asigna una trayectoria agrupadas en la matriz $M_{|U|\times|T|}$, donde cada celda m_t^i representa el movimiento realizado por el UAV u_i en el instante de tiempo t. Para determinar la ubicación de u_i en el instante t, se recorre toda la fila m^i aplicando las modificaciones de ubicación a partir de la posición inicial (b_x, b_y).

Obstáculos. Dentro del área de vigilancia, se encuentran $|O|$ obstáculos representados con forma rectangular que se deben evitar. Estos obstáculos son elementos como lagos, árboles o torres en un entorno real. Cada obstáculo o_i $(0 \leq i \leq |O|)$ se define mediante dos coordenadas cartesianas, que indican dos esquinas opuestas del obstáculo.

POI. Dentro del área de vigilancia, se encuentran $|P|$ puntos de interés que tienen una prioridad de vigilancia mayor que el resto del terreno. Cada punto de interés p_i $(0 \leq i < |P|)$ se define mediante dos valores:

1. Sus coordenadas, representadas como un punto (x, y) en el área. Para alta definición el punto se define como (x^H, y^H)
2. El tiempo mínimo de visita, representado como v_i. Se busca que cada punto de interés sea visitado al menos una vez cada v_i unidades de tiempo.

Métricas de calidad. Para comparar diferentes trayectorias dentro del área de vigilancia es necesario contar con métricas que asignen una calidad a dichas trayectorias. Las métricas consideradas para este proyecto son las siguientes:

1. Puntos de interés visitados: debido a que cada punto de interés tiene un tiempo de espera entre visitas, se calcula la métrica de vigilancia como el porcentaje total de tiempo durante el cual los puntos de interés se encuentran en estado de necesidad de visita. Esta métrica devuelve el puntaje máximo cuando ningún punto de interés pasa una unidad de tiempo necesitando ser visitado.

2. Área de vigilancia cubierta: al finalizar las trayectorias de los UAVs, se cuenta la cantidad de sectores cubiertos en el área de vigilancia. Aquellos sectores que no fueron vigilados se penalizan de manera uniforme. Esta métrica devuelve el puntaje máximo si al finalizar el tiempo se cubrieron todos los sectores al menos una vez.

3. Colisión entre UAVs: se realiza un cálculo que penaliza la presencia de dos o más UAVs en un mismo sector, con el objetivo de minimizar posibles colisiones y, a su vez, maximizar la eficiencia del espacio utilizado para la vigilancia. Esta métrica devuelve el puntaje máximo si nunca hubo más de un UAV en un mismo sector al mismo tiempo.

4. Evitación de obstáculos: se busca minimizar el paso de los UAVs por áreas que contienen obstáculos. Se calcula el tiempo en el que los UAVs sobrevuelan áreas no deseadas. Esta métrica devuelve el puntaje más alto si ningún UAV sobrevoló áreas no deseadas en ningún momento.

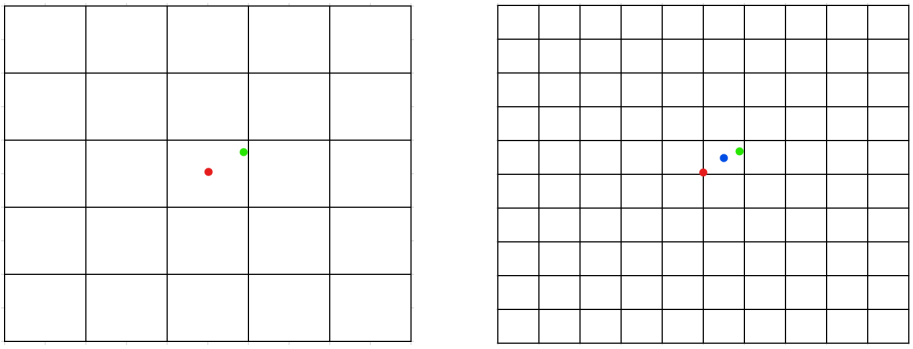
5. Permanencia dentro del área: se busca que los UAVs permanezcan dentro del área de vigilancia durante toda la trayectoria. Esta métrica se utiliza para penalizar a los UAVs segun vuelan por fuera del área de vigilancia, y según la distancia máxima al área que alcanzan.

Al componente diseñado para el calculo de todas estas métricas se lo denomina evaluador. El evaluador se programa de forma modular, esto implica que llegado el caso se pueden agregar o quitar módulos para adaptar el evaluador al contexto deseado. Cada una de estas métricas se encuentra en el rango [0.,1] donde un valor más alto indica una mejor evaluación.

4.2. Problema a resolver

El problema abordado en este proyecto es la generación de trayectorias para UAVs en un área bidimensional, con el objetivo de maximizar su rendimiento al cubrir POIs, evitando obstáculos y considerando la carga de la batería. Específicamente, se busca mejorar la resolución de las trayectorias. La mejora de la resolución es crucial, ya que, al moverse el UAV por los centroides, una baja resolución puede resultar en una vigilancia ineficaz de los POIs definidos, debido a que el UAV se encuentra demasiado alejado del POI. En contraste, un aumento en la resolución permite que el UAVs se posicione más cerca de los POIs (ver Figura 4.1)

Se busca utilizar métodos alternativos a los algoritmos greedy para la generación de trayectorias. Una de las tecnologías emergentes en la generación de datos son las GANs, las cuales son investigadas y empleadas con el objetivo de resolver el problema planteado.



(a) Grilla de baja resolución con POI en verde y centroide del cuadrante correspondiente al POI en rojo

(b) Grilla de alta resolución con POI en verde y centroide del cuadrante correspondiente al POI, en rojo baja definición y en azul alta definición

Figura 4.1: Diferencia de resoluciones

4.3. Generación de trayectorias

Las trayectorias generadas por los algoritmos requieren de parámetros específicos para llevar a cabo su generación. A continuación, se enumeran los parámetros necesarios:

1. Discretización del terreno de vigilancia representada como una dupla de números enteros dim_x, dim_y.
2. Duración de la vigilancia representada como un numero entero T.
3. Cantidad de UAVs que sobrevolarán el terreno expresado como un número entero.
4. Duración de la batería de los UAVs en unidades de tiempo representado como un número entero T_v.
5. Unidades de tiempo requeridas para cargar la batería de un UAV expresado como un entero T_r.
6. Ubicación del punto de partida/base de los UAVs representado como una dupla de números enteros (b_x, b_y).
7. Ubicaciones de los puntos de interés representado como un set de puntos P.
8. Tiempos de visita de cada punto de interés representado como un set de números enteros V.
9. Probabilidad de dirigirse a un punto de interés.

10. Ubicación de los extremos de los obstáculos, expresado como un set de duplas de coordenadas cartesianas O.

11. Penalización de la probabilidad de elegir un movimiento que se dirige hacia un obstáculo expresado como una probabilidad, donde 100 % prohíbe los movimientos sobre obstáculos.

Los puntos 1, 2, 4, 5, 6, 7, 8 y 10 tienen dos representaciones posibles, una para alta resolución a la cual se la nombra como el nombre de la variable supra H y una para baja resolución la cual se la nombra como nombre de la variable supra L.

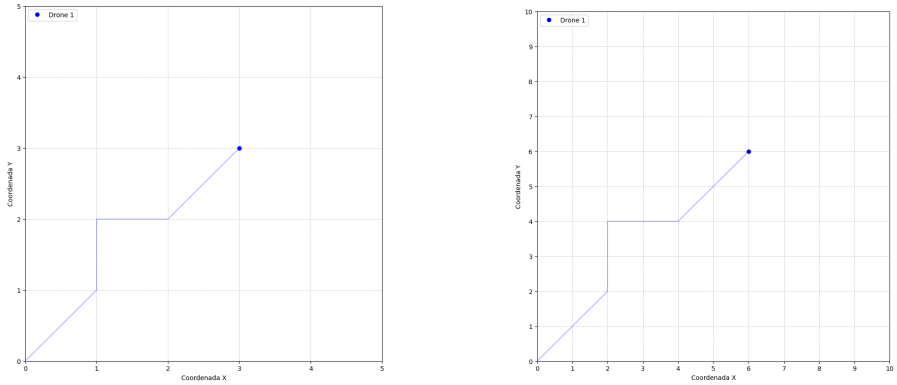
Para generar trayectorias de baja resolución, se implementa un algoritmo greedy probabilístico que produce trayectorias. A continuación se describe el algoritmo: inicialmente, todos los UAVs comienzan en estado de "deambulación" (wandering), recorriendo con igual probabilidad todos los movimientos disponibles. Si un UAV encuentra un obstáculo, se le asigna una penalización a la probabilidad de elegir un movimiento que lo lleve hacia el obstáculo. Cuando un punto de interés debe ser visitado, se asigna al UAV correspondiente y se incrementa la probabilidad de elegir movimientos que lo acerquen al punto de interés. Cada vez que un UAV realiza un movimiento, se calcula la cantidad de tiempo necesaria para llegar a la base y se compara con el tiempo restante de vuelo del UAV. Si los tiempos son iguales, el UAV se dirige a la base evitando obstáculos en su trayectoria.

4.4. Mejora de resolución

El objetivo principal de este proyecto es evaluar la posibilidad de utilizar una GAN para aumentar la resolución de los resultados obtenidos por el algoritmo greedy descripto en la sección 4.3. El aumento de resolución implica incrementar las dimensiones de la grilla que representa el área de vigilancia y la cantidad de unidades de tiempo que dura la vigilancia por un factor. La diferencia en unidades de tiempo no representa mayor o menor cantidad de tiempo, sino subdivisiones de tiempo mayores o menores tales que permita representar la mayor precisión en los movimientos. En la figura 4.2 se muestra una trayectoria representada en baja y alta resolución. Si bien la trayectoria es la misma, en el caso de alta resolución permite una mayor precisión al momento de realizar los movimientos que la componen.

En la Figura 4.3 se muestra la arquitectura de la GAN utilizada para la generación de trayectorias de alta resolución. El dataset generado mediante la heurística greedy se emplea como una de las entradas del discriminador. El generador recibe como entrada ruido y produce una trayectoria de UAVs de alta resolución. La trayectoria generada se evalúa utilizando las métricas indicadas en la sección 4.1. Posteriormente, la trayectoria generada se ajusta a la resolución utilizada por la heurística para ser procesada por el discriminador. El discriminador, a su vez, proporciona un resultado que se emplea para su propio aprendizaje y para la mejora del generador.

Las trayectorias generadas por el componente generador de la GAN son de alta resolución. Estas trayectorias son luego re-escaladas a baja resolución, resolución utilizada por el algoritmo greedy, mediante un componente denominado downscaler.



(a) Trayectoria en baja resolución (b) Trayectoria en alta resolución

Figura 4.2: Misma trayectoria en baja y alta resolución

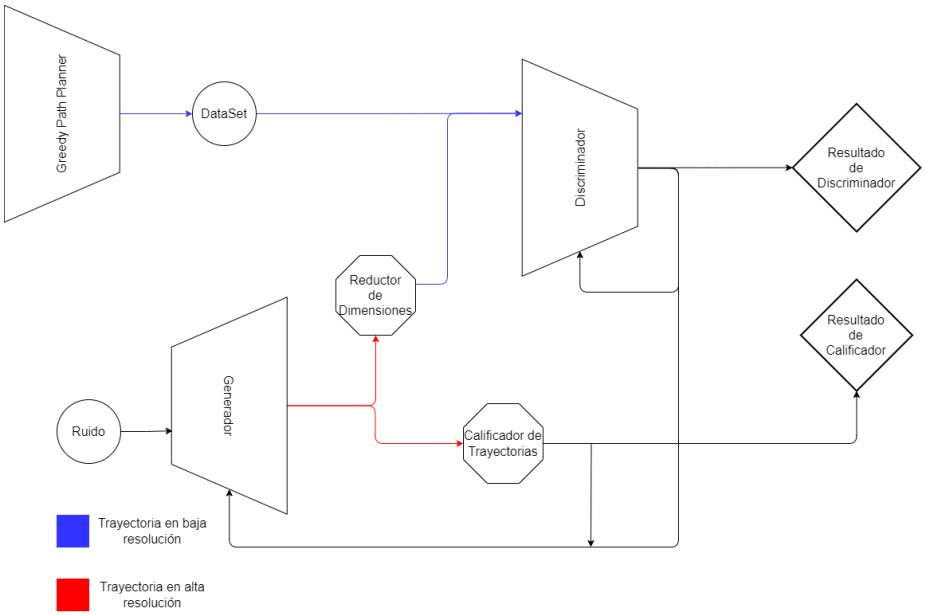


Figura 4.3: Arquitectura de la GAN

Capítulo 5

Desarrollo incremental y validación

En este capítulo se detalla la implementación de un algoritmo greedy para la generación de trayectorias de UAVs, se presentan y analizan los resultados obtenidos al aplicar múltiples arquitecturas de GANs en la generación de trayectorias de UAVs para la vigilancia de áreas.

5.1. Generación de trayectorias greedy

Para generar el dataset que luego se utiliza como entrada para el entrenamiento de la GAN se desarrolló e implementó un algoritmo greedy capaz de generar trayectorias de UAVs en un contexto definido tal como es detallado en el capítulo 4. Este algoritmo genera trayectorias que visitan a los POIs definidos, intentando evitar obstáculos y cargando la batería cuando sea necesario antes de continuar con su plan de vuelo.

La salida del algoritmo (ver anexo A.1) se compone de tres líneas en un archivo de texto, en donde cada línea representa a un UAV, y cada número una acción de las definidas en el capítulo 4. La figura 5.1 muestra una trayectoria generada por el algoritmo greedy utilizando PyPlot (Hunter, 2007) para representarla. Estas trayectorias se generaron utilizando una heurística aplicada a cada UAV, de manera que ningún UAV tiene información del resto de los UAVs. Mientras no haya necesidad de la vigilancia de un POI, el algoritmo toma decisiones de forma aleatoria para mover el UAV y al momento de tener la responsabilidad de visitar un POI, el UAV se dirige hacia el POI.

Para permitir cambiar rápidamente la heurística o desarrollar nuevas, se llevó a cabo la implementación de forma modular. La implementación modular significa que la función que se llama para obtener el nuevo estado de un UAV se define en un módulo separado con ciertas funciones base establecidas. El desarrollo modular permite implementar soluciones alternativas a la inicialmente desarrollada para distintos objetivos. Para verificar esta funcionalidad, se im-

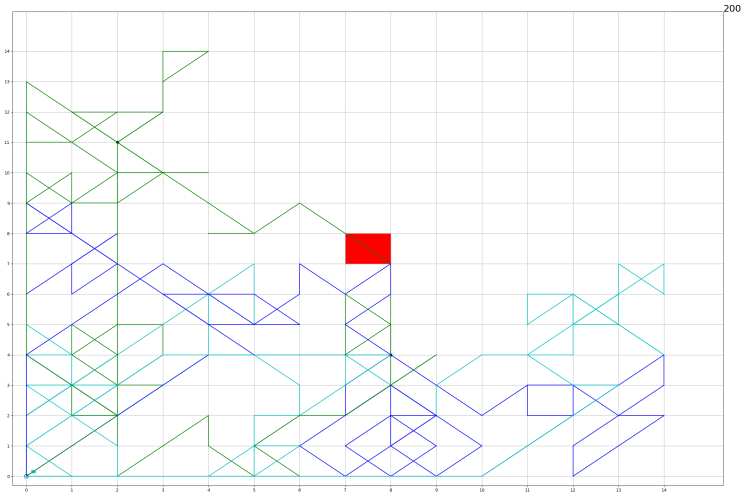


Figura 5.1: Trayectoria generada con el algoritmo greedy para tres UAVs, dos POIs y un obstáculo.

plementó la heurística llamada Nefesto, que para cada UAV, en cada instante de tiempo, calcula los posibles movimientos y siempre toma uno aleatorio. El comportamiento de la heurística Nefesto contrasta con la heurística por defecto, Ardemisa, que mantiene este comportamiento aleatorio hasta que se requiere vigilar un POI, momento en el cual se dirige directamente a vigilarlo.

5.2. GAN sin evaluador ni aumento de resolución

Se definió una GAN convolucional siguiendo estructuras ampliamente conocidas para el entrenamiento de datasets como MNIST y CelebA. Las arquitecturas están compuestas por capas convolucionales, funciones de activación LeakyReLu y una activacioón final sigmoide. Dado que esta arquitectura trabaja con valores entre 0 y 1, el dataset se normalizó a este rango para asegurar la compatibilidad con el Generador y el Discriminador.

El entrenamiento de la GAN comenzó sin la utilización de la función de evaluación ni el aumento de resolución. Esta decisión se tomó con el objetivo de superar problemas de forma incremental. La estrategia consiste en abordar y resolver problemas de menor complejidad inicialmente, y luego incrementar gradualmente la complejidad del problema. De esta manera, se facilita un enfoque más simple y específico para abordar los desafíos posteriores.

El dataset utilizado consta de 1000 ejemplos generados por el algoritmo greedy. El modelo se entrenó durante 5000 épocas, obteniéndose los resultados visibles en las figuras 5.2 y 5.3. Estas figuras muestran que la función de pérdida (loss) del discriminador disminuye conforme avanzan las épocas, mientras que la del generador aumenta. Aunque el objetivo es que ambas pérdidas disminuyan, este comportamiento es esperado. Este comportamiento se debe a que, a medida que el discriminador mejora en distinguir entre imágenes generadas y reales, la pérdida del generador aumenta al recibir retroalimentación más precisa sobre sus fallos.

La figura 5.3 muestra las trayectorias generadas. Las trayectorias se asemejan a trayectorias reales pero presentan un problema que se consideró grave, los UAVs permanecen más de un 30 % del tiempo fuera del área definida. Otros problemas importantes que se destacan son el no volver a la base de carga, permanecer estático en un lugar por mucho tiempo y no visitar los POIs definidos en el área. En el contexto definido, estas trayectorias son lo suficientemente deficientes como para ser descartadas. Esto indica que la red no logró captar adecuadamente la necesidad de recarga de batería, moverse por el área, visitar los POIs y no salirse del área.

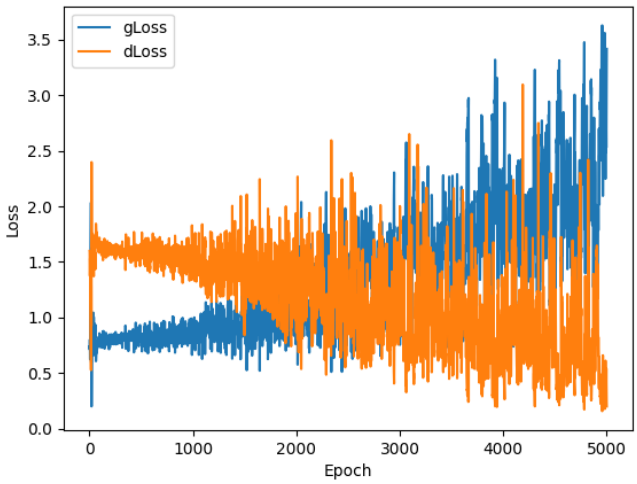


Figura 5.2: Pérdida de GAN sin evaluador en 5000 épocas

5.3. One Hot Encoding

Se realizaron múltiples iteraciones sobre la estructura de la red neuronal y la forma en que los datos son procesados. Una de estas iteraciones consiste en la representación de los datos mediante One Hot Encoding (OHE). En esta

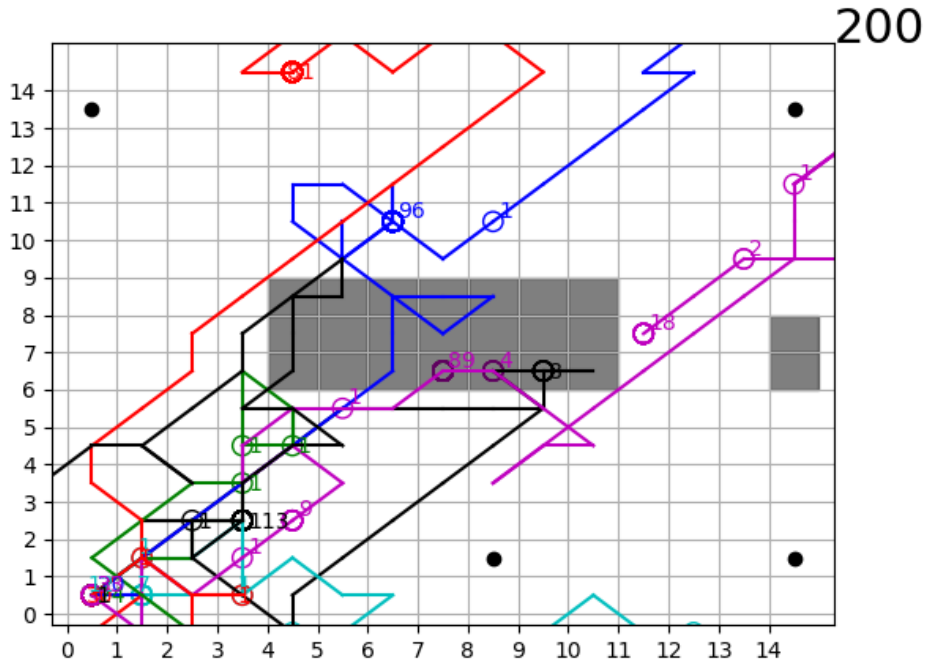


Figura 5.3: Trayectoria generada por la GAN sin evaluador entrenando 5000 épocas

representación, los movimientos, en lugar de ser representados numéricamente con un valor entre cero y ocho, se representan mediante un vector de ocho elementos. Cada número n se representa como un vector cuyo n-ésimo elemento tiene valor uno y el resto de los elementos son cero. En el caso de la acción de no movimiento (representada con un cero) es codificada como un vector de ocho ceros. Esta reestructuración de los datos implicó una reestructuración de la GAN, particularmente para permitirle procesar datos con las nuevas dimensiones.

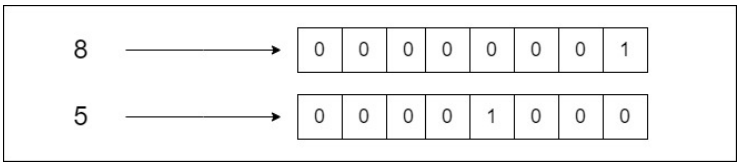


Figura 5.4: Representación OHE

La experimentación se realizó con el objetivo de evaluar la capacidad de la GAN para interpretar correctamente los datos y obtener una métrica de distancia entre acciones de UAVs que sea fiel a la realidad. En particular, se observó que la distancia entre acciones similares no está bien representada. Por ejemplo, las acciones “DERECHA” y “DIAGONAL SUPERIOR DERECHA”

son conceptualmente próximas, sin embargo, los números que las representan, “1” y “8”, están alejados entre si. OHE permite representar a todos los distintos movimientos como opciones distintas, “DERECHA”, “DIAGONAL SUPERIOR DERECHA” y “DIAGONAL INFERIOR IZQUIERDA” todas tienen la misma distancia entre si. Un problema de esta representación es que, si bien soluciona el problema de la incorrecta representación de la distancia entre acciones similares, no permite que los movimientos conceptualmente próximos tengan una menor distancia en comparación con movimientos conceptualmente alejados.

Los resultados producidos por la GAN no fueron utilizables debido a que la red no produjo vectores con un único elemento con valor uno. Se modificó entonces al generador para que haga uso de la capa de activación softmax, que reescala la salida del generador para que los elementos de los vectores deban sumar uno en total, y tomen un valor entre cero y uno. Haciendo uso de la capa softmax se generaron trayectorias con la GAN entrenando con el set de datos en formato OHE. Los resultados obtenidos muestran una tendencia de los UAVs a llevar a cabo las trayectorias fuera del área de vigilancia. La figura 5.5 muestra una trayectoria generada por la red, donde el área de vigilancia se encuentra marcada con negro y la trayectoria del UAV en azul.

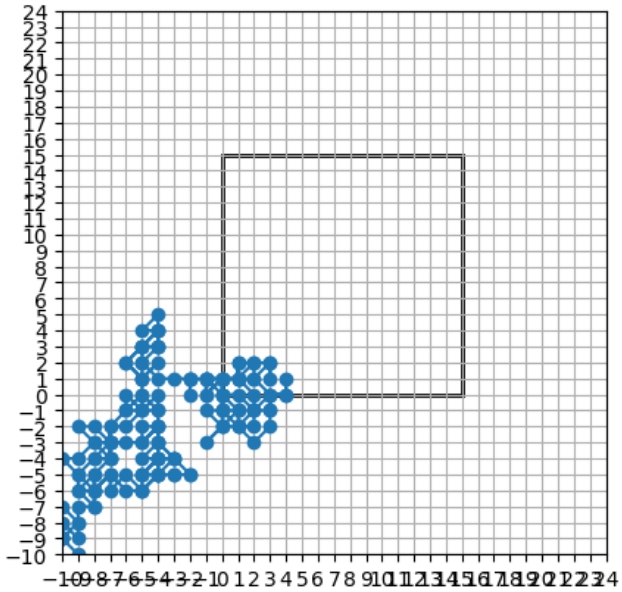


Figura 5.5: Trayectoria generada por la GAN, usando codificación OHE con 500 épocas de entrenamiento utilizando activación softmax

5.4. Personalización de la función de pérdida

En esta sección se documenta la inclusión de una función de pérdida modificada al ciclo de entrenamiento de la GAN y los resultados obtenidos.

5.4.1. Inclusión de función de pérdida en el entrenamiento

La función de pérdida estándar de TensorFlow compara la salida del discriminador con el valor esperado. En contraste, la función de pérdida personalizada permite evaluar y guiar al generador de la GAN en función del puntaje conseguido al evaluar las trayectorias generadas según las métricas definidas en la sección 4.1. Este enfoque tiene como objetivo dirigir al generador para que produzca trayectorias que minimicen las salidas del área designada.

Debido a las limitaciones y problemas encontrados con TensorFlow al personalizar la función de pérdida del generador, se transicionó a PyTorch. En este nuevo entorno, se implementó un perceptrón multicapa, en contraste con el modelo anterior que utilizó capas convolucionales. En una primera instancia, para verificar la correcta implementación en PyTorch se ejecutó un entrenamiento de la red sin intervención del evaluador. Luego de experimentar con varias combinaciones de hiperparámetros sin obtener trayectorias factibles, se optó por simplificar el problema a un único UAV, requiriendo un ajuste en la entrada de la red. Se ejecutó el algoritmo greedy para generar un conjunto de datos de tamaño 4000 con un único UAV y 200 unidades de tiempo. Nuevamente se realizaron experimentos con el perceptrón multicapa sin el evaluador obteniendo resultados no factibles, un ejemplo de estos resultados se muestra en la figura A.1 en donde el UAV sale del área de vigilancia y no regresa.

Se utilizó la entropía cruzada binaria (BCE, por sus siglas en inglés) como función de pérdida en la GAN, comparando las salidas esperadas del discriminador (unos para las trayectorias generadas por el generador) con las obtenidas. Además, para incorporar la evaluación de las trayectorias en esta función de pérdida, se implementó una ponderación entre la función de pérdida estándar y la función de evaluación. La función de pérdida personalizada se muestra en el anexo A.2.

5.4.2. Experimentación y resultados

Se realizó un primer entrenamiento con las tasas de aprendizaje para el generador y el discriminador en 1×10^{-4} y se lo entrenó por 1000 épocas. Los resultados son visibles en la figura 5.6. A partir de estos resultados se decidió que en los consecuentes experimentos se utilizara 400 épocas ya que la gráfica muestra como a partir de de época 390 aproximadamente comienzan a separarse nuevamente las pérdidas de ambas componentes de la GAN. A su vez en la figura 5.6 se muestra como en un principio los valores de la evaluación de trayectorias tienen una tendencia de crecimiento, convergiendo a un valor cercano a 0.4.

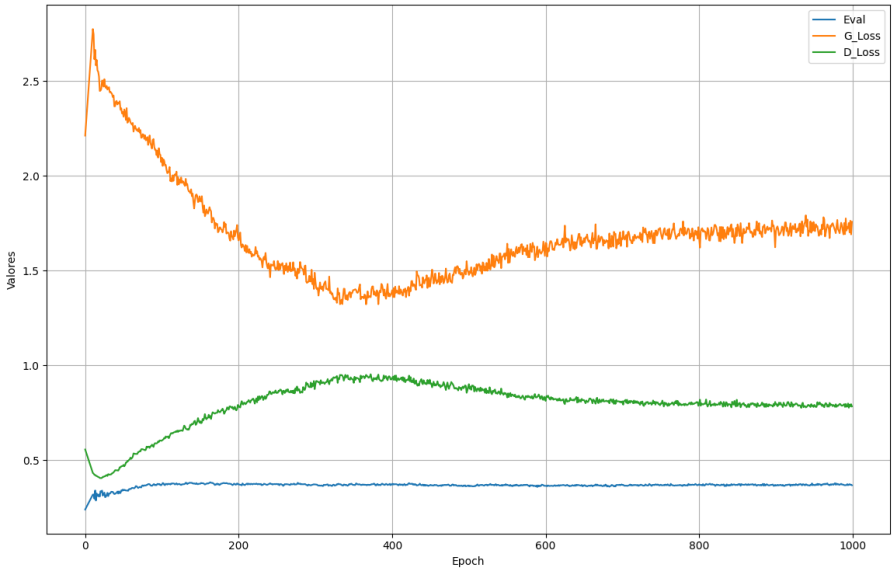


Figura 5.6: Gráfica de las funciones de pérdidas y la salida del evaluador utilizando pesos constantes

Los resultados obtenidos para algunos de los experimentos se encuentran en la tabla 5.1, donde GLR es el learning rate utilizado por el generador, DLR es el learning rate utilizado por el discriminador, EW es la ponderación utilizada para el evaluador, RW es la ponderación utilizada para la funcion BCE, D Loss es la pérdida del discriminador, G Loss es la pérdida del generador y Eval es el promedio de la evaluación realizada por el evaluador sobre las trayectorias generadas. No hubo una variación grande en las evaluaciones, se mantienen por debajo de 0.4, lo que se considera no satisfactorio. En la figura 5.7 se muestra la representación de la trayectoria de la salida para el caso del learning rate de 1×10^{-4} y 400 épocas. En esta imagen se muestra que el UAV si bien sale del área no se va muy lejos de esta, sin embargo no hubo ninguna instancia de tiempo en donde el UAV haya pasado por alguno de los POIs definidos, lo que la descalifica como solución satisfactoria. El promedio de permanencia dentro del área estipulada es de 2.18 %.

Con el objetivo de que el generador genere trayectorias factibles y que se mantengan dentro del perímetro de vigilancia, se implementó una estrategia de técnica de evolución de los pesos. En este enfoque, la función de pérdida de la red es una suma ponderada entre la evaluación de las trayectorias generadas y la entropía binaria cruzada entre la salida del discriminador y el valor esperado. La ponderación varía de acuerdo a las épocas transcurridas del entrenamiento. La función de pérdida de la red se representó con la siguiente fórmula: $L_k(g) = (1 - k/M)BCE(D(g), 1) + (k/M)BCE(E(g), 1)$ donde L_k es la función de pérdida

Cuadro 5.1: Resultados de la GAN con 400 épocas para la experimentación con pesos constantes en la función de pérdida personalizada

GLR	*DLR*	*EW*	*RW*	*D Loss*	*G Loss*	*Eval*
1×10^{-4}	1×10^{-5}	0.2	0.8	1.237	0.734	0.368
1×10^{-4}	1×10^{-5}	0.4	0.6	1.259	0.541	0.391
1×10^{-4}	1×10^{-4}	0.2	0.8	1.077	1.225	0.371

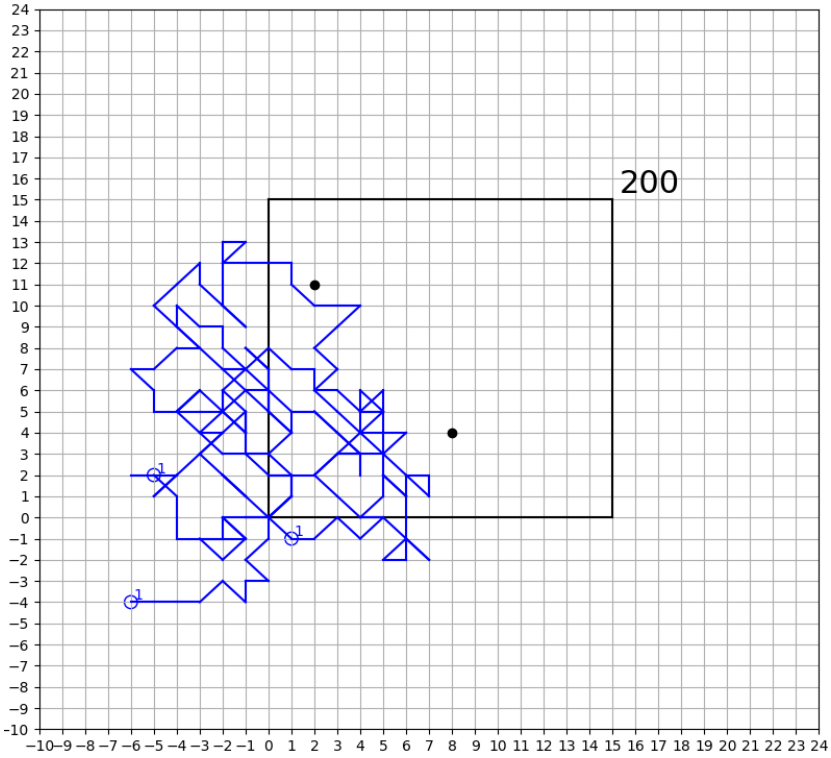


Figura 5.7: Trayectoria generada por la GAN con DLR 1×10^{-4} en la época 400 con pesos constantes

para la época k, g es la salida generada en la época k, BCE es la función de entropía binaria cruzada, $D(g)$ es la salida del discriminador para el vector generado g , M es el numero de épocas definido y $E(g)$ es la salida del evaluador. Esta función es lineal y se experimentó con diferentes hiperparámetros. La figura del anexo A.2 muestra la representación de una trayectoria obtenida utilizando la ponderación lineal. Esta representación muestra como la mayor parte de la trayectoria se lleva a cabo fuera del área de vigilancia. Se obtiene un promedio de 0.65 % de permanencia dentro del área estipulada con este enfoque. Los

resultados obtenidos son inferiores a los obtenidos con los pesos constantes por un factor de tres.

Se realizaron experimentos con una función exponencial para la variación de los pesos, las cuales proporcionaron resultados muy similares e incluso inferiores a los reportados por la función lineal. Si bien la gráfica que se muestra en la figura 5.8 muestra una mejora a lo largo de la épocas la evaluación permanece por debajo de 0.5 en todo momento y se mantiene estable sin mejoras. Las trayectorias generadas (ejemplo en la figura del anexo A.3) continúan saliéndose del área, permaneciendo en promedio un 0.51 % del tiempo de vuelo dentro del área y la evaluación de las trayectorias es peor a la de las trayectorias generadas utilizando los pesos constantes e incluso a la evolución lineal.

Al comparar los tres enfoques, el porcentaje promedio de permanencia dentro del área fue mayor con el uso de pesos constantes, alcanzando un 2.18 %. En contraste, el enfoque de evolución lineal registró un porcentaje de 0.65 %, mientras que la evolución exponencial mostró el valor más bajo con un 0.51 %. Si bien los resultados provisto por el enfoque constante no son los esperados, los enfoques dinámicos, lineal y exponencial no solo no proveen una mejora sino que hay un notorio empeoramiento en los resultados.

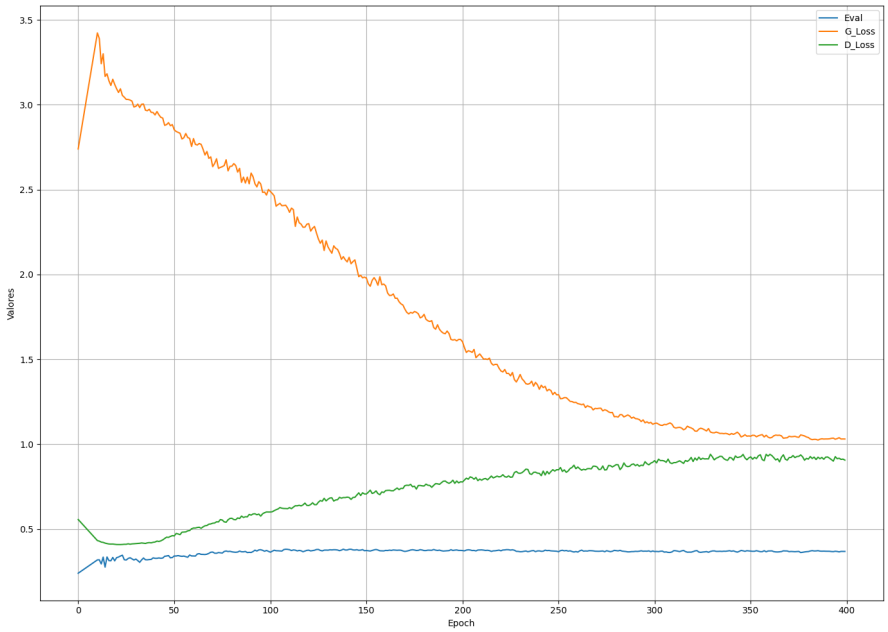


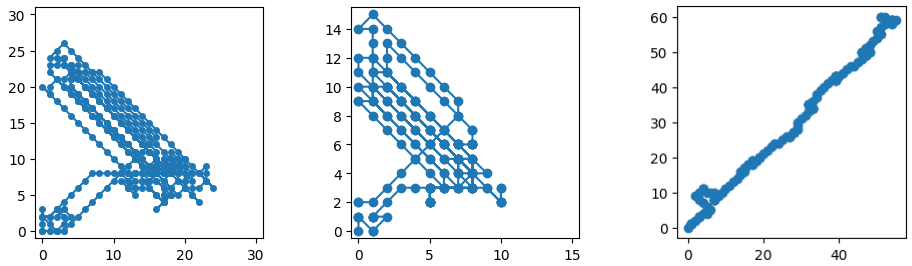
Figura 5.8: Técnica de evolución de los pesos con función exponencial de actualización de los pesos

5.5. Introducción del Downscaler

El objetivo de la GAN es la generación de trayectorias con mayor grado de detalle que las originalmente provistas. Para que el entrenamiento pueda llevarse a cabo es necesario que lo generado por la GAN sea comparable con los datos utilizados para su entrenamiento, lo que no es posible si existe una diferencia de dimensiones entre ambos conjuntos de datos. Para esto es necesario la implementación de un módulo que adapte las trayectorias de alta resolución a su equivalente en baja resolución, al cual se lo denomina downscaler. La implementación del downscaler esta ligada a la representación de las trayectorias. Para el caso de trayectorias representadas como acciones, la función del downscaler esta dada por 5.1 donde T^H es la trayectoria en alta resolución, T^L es la trayectoria en baja resolución, T_i es el i-ésimo elemento de una trayectoria y $\delta(T)$ es la variación en las coordenadas (x, y) generadas por la acción T.

$$\delta(T_i^L) = \lceil \frac{\delta(T_{2i}^H) + \delta(T_{2i+1}^H)}{2} \rceil \tag{5.1}$$

La introducción del downscaler al ciclo de entrenamiento de la red resultó en pérdida absoluta de aprendizaje debido a que las funciones propuestas no mantienen los gradientes necesitados por el framework Pytorch. Para poder mantener los gradientes de los tensores se desarrolló una red neuronal perceptrón multicapa, con la función de diferencia cuadrática media como pérdida, para que aprenda la función del downscaler. Esta implementación del downscaler brindó resultados no útiles tal como se muestra en la figura 5.9c. Estos resultados en parte se deben al cálculo de pérdida entre dos acciones, ya que el uso de diferencia cuadrática media, no representa correctamente las diferencias entre distintas acciones. Un ejemplo de esto es como las acciones Derecha y Diagonal Superior Derecha, son similares, pero su diferencia mediante la función de pérdida es mayor a la del resto de las acciones entre si. Esta experiencia llevó a la reestructuración de la representación de las trayectorias.



(a) Trayectoria de alta resolución

(b) Baja de resolución mediante la función de entrenamiento

(c) Baja de resolución mediante la red neuronal

Figura 5.9: Comparación de baja de resolución esperada y obtenida

5.6. Recodificación del dataset: representación cartesiana

En esta sección se detallan los objetivos de la recodificación a representación cartesiana, el método utilizado para la recodificación de las trayectorias, el impacto que esta nueva codificación de las trayectorias tiene sobre la arquitectura y los resultados que se obtuvieron.

5.6.1. Objetivos de la recodificación

Con la nueva representación cartesiana se busca limitar al generador a generar trayectorias que estén únicamente dentro del área. Al utilizar una arquitectura de la GAN en donde la capa final del generador es una capa de activación sigmoide, la cual limita la salida entre cero y uno y al multiplicar este valor por el máximo de la dimensión del área y redondearlo se obtienen trayectorias exclusivamente dentro del área estipulada.

La codificación de trayectorias por acciones genera una fuerte dependencia de todas las acciones con las primeras acciones de la trayectoria. En caso que las primeras acciones de una trayectoria lleven al UAV hacia fuera del área, entonces toda la trayectoria se lleva a cabo fuera del área. Esta dependencia no está presente en codificaciones mediante coordenadas cartesianas, donde cada coordenada de la trayectoria solo es dependiente de sus adyacentes para no llevar a cabo un desplazamiento no viable por un UAV.

5.6.2. Recodificación

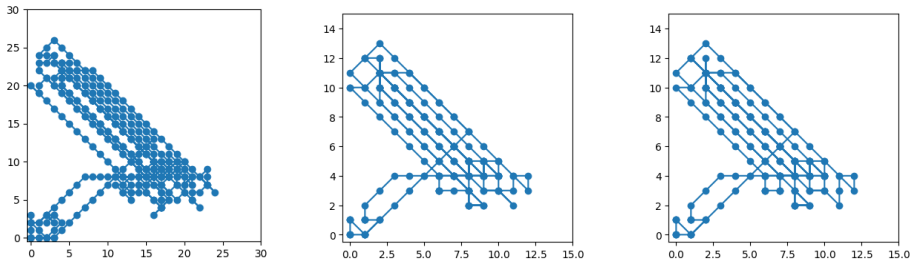
Se implementó un script que toma el dataset original con acciones y lo transforma en un dataset que representa las trayectorias como una secuencia de puntos cartesianos, aplicando secuencialmente los deltas generados por las acciones a la coordenada de origen. Si la secuencia de acciones es ARRIBA DERECHA, cuyos deltas son (0,+1) y (+1,0) respectivamente se transforma a [(0,0),(0,1), (1,1)].

El método de recodificación que se utilizó garantiza que la trayectoria es equivalente a la original y que cada coordenada cartesiana es adyacente a la previa en la secuencia que define la trayectoria. A todo par de coordenadas continuas en una trayectoria, no adyacentes en el plano en cualquiera de las ocho direcciones en las que se permiten los movimientos, se refiere como un salto.

5.6.3. Impacto en la arquitectura

Esta nueva representación cambió las dimensiones necesarias para representar una trayectoria. Una trayectoria que mediante acciones consiste en una lista de T elementos singulares, se representa en coordenadas cartesianas como una lista de T duplas. Esto requirió de un cambio en las dimensiones del generador así como el discriminador para admitir los nuevos tamaños (ver anexo A.4.1).

La función que se utiliza para entrenar el módulo de downscaling debe contemplar la nueva representación de las trayectorias. Se cambió la función de downscaling por $T_i^L = \frac{T_{2i}^H + T_{2i+1}^H}{2}$. Se entrenó nuevamente el downscaler con la nueva función de downscaling, y se logró exitosamente obtener una red capaz de llevar a cabo una baja de resolución tal como se muestra en la figura 5.10c.



(a) Trayectoria de alta resolución

(b) Baja de resolución mediante la función de entrenamiento

(c) Baja de resolución mediante la red neuronal

Figura 5.10: Comparación de baja de resolución esperada y obtenida para la representación cartesiana

5.6.4. Resultados

Se ejecutaron varios experimentos con distintos hiperparámetros en la búsqueda de un entrenamiento estable. La primer experimentación se realizó con ambos learning rates configurados en $1{\times}10^{-4}$. El entrenamiento inmediatamente presentó problemas tal como lo muestra la imagen 5.11. El discriminador rápidamente logró diferenciar con certeza imágenes reales de las generadas por el generador lo cual no le permitió al generador entrenarse. Para solucionar este problema se analizaron varias combinaciones de learning rates, aumentando de a $1{\times}10^{-4}$ el learning rate del generador y reduciendo por un factor de diez cada vez el learning rate del discriminador hasta lograr una combinación de $5{\times}10^{-4}$ para el generador y $1{\times}10^{-6}$ para el discriminador que presentó una evolución estable a lo largo de las épocas, esta evolución se muestra en la figura 5.12. El entrenamiento se detuvo en la época 1580 donde se encontró que las losses se encontraban divergiendo. Se presentó nuevamente el problema de que el discriminador mejora al punto que el generador deja de aprender. La figura 5.13 muestra la trayectoria generada por este último entrenamiento. En la trayectoria se representan los saltos como lineas rojas y los puntos de partida y llegada del salto como puntos rojos en forma de X . Los movimientos validos son representados como lineas azules. La figura 5.13 muestra como el generador no logró aprender la lógica de la generación de trayectorias con la mayoría de sus movimientos siendo no validos.

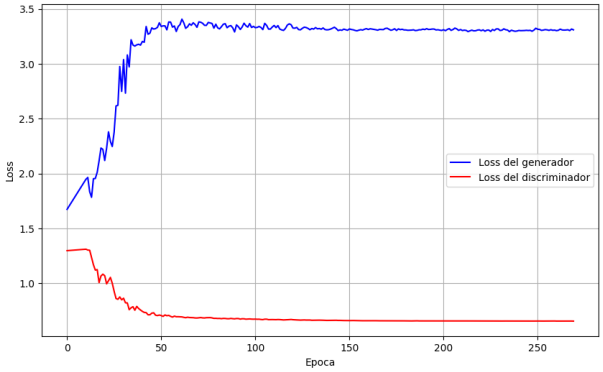


Figura 5.11: Loss del generador y discriminador a lo largo de las épocas con GLR y DLR de 1×10^{-4}

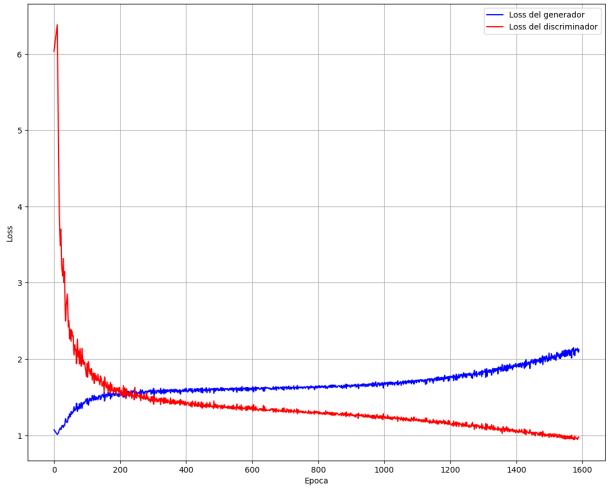


Figura 5.12: Loss del generador y discriminador a lo largo de las épocas con GLR de 5×10^{-4} y DLR de 1×10^{-6}

5.7. LSTM GAN

Debido a la naturaleza secuencial de los datos se implementó y experimentó con una LSTM-GAN. Dicha implementación tuvo varias configuraciones en las cuales se experimentó con varias arquitecturas similares. La configuración que logró mejores resultados fue con el generador utilizando capas LSTM mientras que el generador se mantuvo con la configuración de perceptrón multicapa (ver anexo A.4.3).

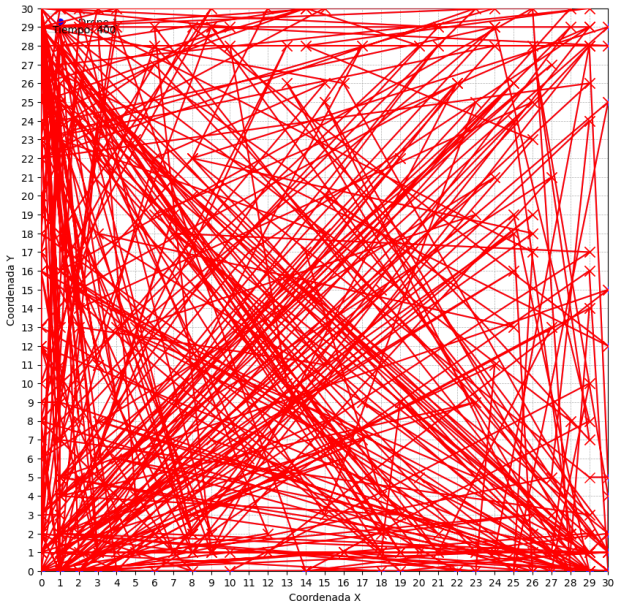


Figura 5.13: Trayectoria generada con la configuración GLR de 5×10^{-4} y DLR de 1×10^{-6} en la epoca 1580

La figura 5.14 muestra la trayectoria generada por el generador de la GAN. En comparación a lo obtenido en la sección 5.6 se obtuvieron trayectorias más estables pero las cuales siguen presentando una gran cantidad de saltos lo cual la hace inviable.

La tabla 5.2 muestra una comparación de la cantidad de saltos promedio presentes en una muestra de 100 trayectorias para la LSTM GAN y para la implementación de perceptrón multicapa. Hay evidencia de una mejora de 80 % en la distancia de los saltos presentes en la trayectoria. Se halla una disminución de 20 % en la cantidad promedio de saltos que se generan en la trayectoria. A pesar de esta mejoría, la cantidad promedio de saltos continúa superior al 50 % de los movimientos de la trayectoria.

Cuadro 5.2: Resumen de saltos promedio de trayectorias generadas por arquitecturas cartesianas de la LSTM GAN en 100 muestras

Caso	*#Saltos*	*Dist. Euclidiana*	*Dist. Manhattan*
Perceptrón GAN	285,07	32,67	36,67
LSTM	226,26	5,98	6,86

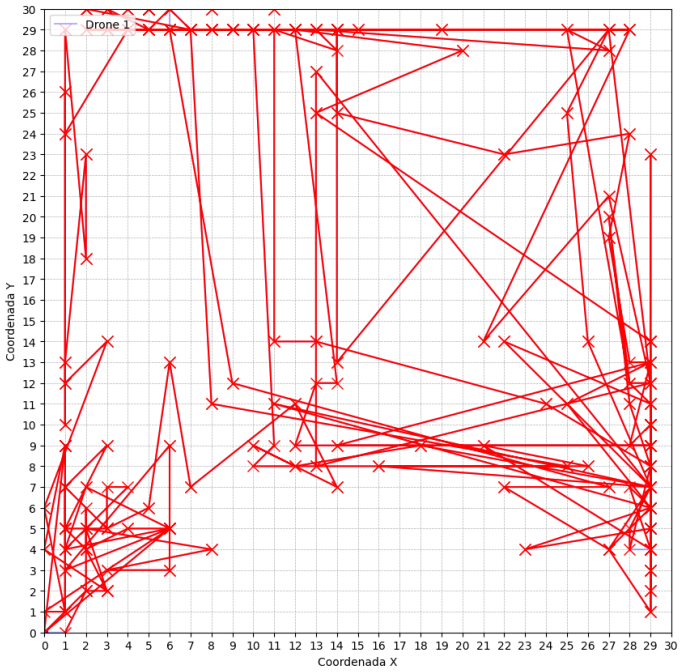


Figura 5.14: Trayectoria generada por red LSTM

5.8. Wasserstein GAN

Se experimentó mediante una adaptación de la red a una arquitectura de Wasserstein GAN (WGAN). Para esto se modificó la función de pérdida utilizada por el discriminador y el generador acorde a la arquitectura WGAN, los nuevo modelos se encuentran en el anexo A.4.2. Tal como indicaron Arjovsky y cols. (2017) el uso de optimizadores basados en momento como Adam o descenso por gradientes con momento, llevan a una mayor inestabilidad del sistema. Debido a esta inestabilidad con optimizadores por momento, se experimenta con la propagación de la raíz cuadrada media (RMSprop) así como con Adam.

Se analizaron los resultados obtenidos mediante la implementación de la WGAN, se encontró una diferencia notable entre los resultados obtenidos con Adam en comparación con los obtenidos por RMSprop. Como muestra la tabla 5.3, el uso de la WGAN con un optimizador RMSprop resultó en un aumento en la cantidad promedio de saltos en las trayectorias generadas de 30 %, pero estos saltos son de una magnitud promedio menor a los presentes en las trayectorias generadas por el perceptrón multicapa por un factor de 55 %. Los resultados generados por la red utilizando el optimizador Adam presentan una disminución en la cantidad promedio de saltos y en la magnitud promedio de los saltos de 30 % y 89 % respectivamente, superando las mejoras obtenidas por la red sin uso de optimizador con momento. La figura 5.15 muestra una trayectoria generada

por la WGAN con momento, la trayectoria continúa teniendo alrededor del 50 % de los movimientos inválidos.

Cuadro 5.3: Resumen de saltos promedio de trayectorias generadas por arquitecturas cartesianas de la WGAN en 100 muestras

Caso	*#Saltos*	*Dist. Euclidiana*	*Dist. Manhattan*
Perceptrón GAN	285,07	32,67	36,67
LSTM	226.26	5.98	6.86
WGAN sin momento	368,86	14,33	16,14
WGAN con momento	197,77	3,73	4,28

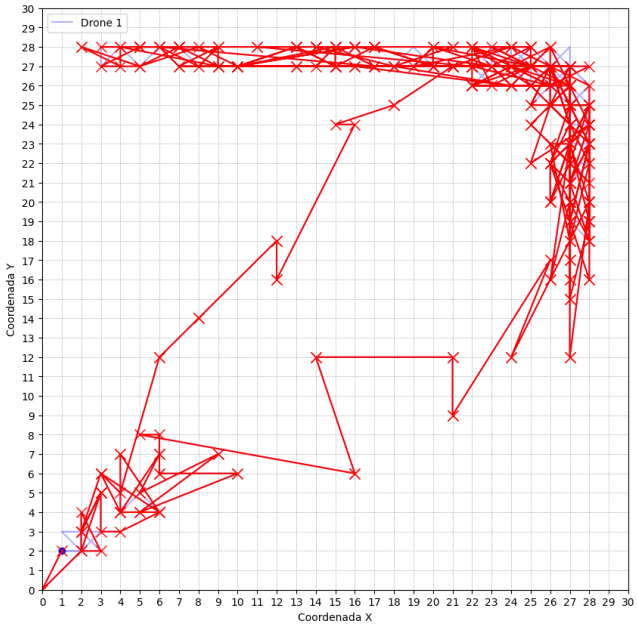
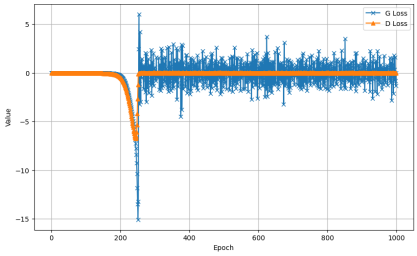
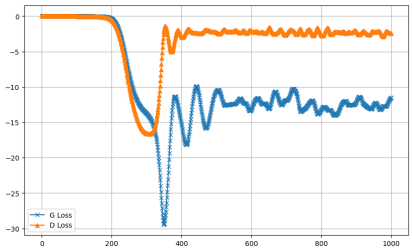


Figura 5.15: Trayectoria generada por la WGAN con 1000 épocas de entrenamiento

Los resultados obtenidos mediante el uso de un optimizador con momento fueron 45 % mejores en cuanto a cantidad promedio de saltos en comparación a la WGAN sin momento. Esto se debe a que la ausencia del momento en la optimización, causa que la red converja a un estado donde no es capaz de generar trayectorias de alta calidad. En cambio el optimizador con momento, permite que la red oscile y alcance estados de capaces de generar trayectorias de calidad. La figura 5.16 muestra como la WGAN sin momento convergió a un estado estable, mientras que el uso de momento no permitió que el generador converja.



(a) Losses de WGAN sin momento



(b) Losses de WGAN con momento

Figura 5.16: Comparación de WGAN con optimizador con/sin momento

5.9. Recodificación del dataset: Representación matricial

En esta sección se detallan los objetivos de la recodificación, el método utilizado para la recodificación de las trayectorias, el impacto que esta nueva codificación de las trayectorias tiene sobre la arquitectura de la GAN y los resultados que se obtienen.

5.9.1. Objetivos de la recodificación

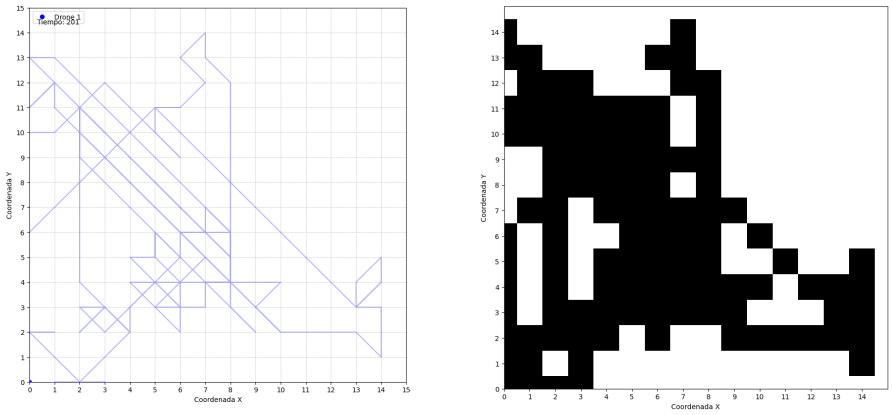
Debido a los saltos que se presentaron en las trayectorias codificadas en coordenadas cartesianas, se buscó una nueva codificación de las trayectorias que no cuente con saltos y que sus movimientos no puedan tener un impacto demasiado fuerte en la trayectoria, como el caso de la codificación por acciones. La codificación a trayectorias matriciales busca eliminar los saltos presentes en las trayectorias cartesianas generadas por el generador. Simultáneamente, al generarse toda el área de vigilancia visitada, se garantiza que las trayectorias se limitan exclusivamente al área de vigilancia, y no fuera de la misma como sucede con la representación por acciones.

Mediante la representación matricial, una trayectoria es interpretada como una imagen. Comprender las trayectorias como imágenes permite experimentar con ellas con mecanismos conocidos para el procesamiento de imágenes mediante redes neuronales, contando con el soporte de las redes convolucionales especializadas en el procesamiento de imágenes.

La corrección de saltos de las trayectorias generadas sin modificar drásticamente la trayectoria original no es posible usando trayectorias codificadas en formato cartesiano. En cambio para la codificación matricial con la que se trabajó, puntos que sean generados disconexos del resto de la trayectoria, pueden ser conectados mediante la adición de puntos a la matriz.

5.9.2. Recodificación

La recodificación del dataset de entrenamiento resultó simple en base al dataset ya existente en coordenadas cartesianas. Se toma cada trayectoria T del dataset, y se transforma en una matriz $M_{dim_x dim_y}$, donde $\forall i \in [0..t], M_{T_i} = 1$. Se obtuvo entonces una recodificación del dataset, con trayectorias sin saltos. En la figura 5.17 se muestra la conversión de una trayectoria codificada mediante coordenadas cartesianas a su equivalente en codificación matricial.



(a) Trayectoria con representación cartesiana

(b) Trayectoria con representación matricial

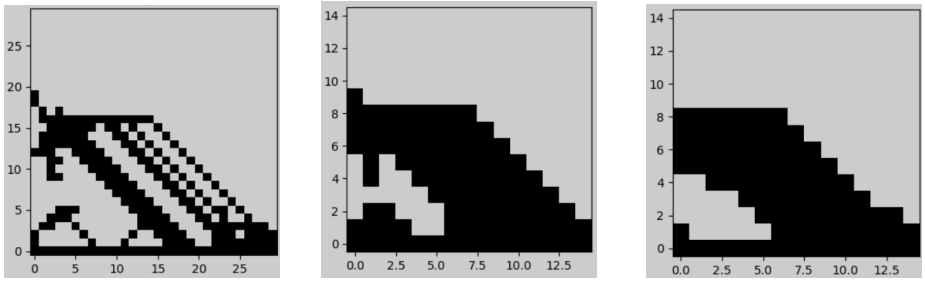
Figura 5.17: Conversión de representación cartesiana a representación matricial

Debido a la pérdida de gradientes documentada en la sección 5.6.2 fue necesaria la implementación del modulo de downscaling mediante una red neuronal. Se implementó la red de downscaling como una red convolucional con tres capas que toma como entrada una trayectoria en alta resolución, y utilizando el error cuadrático medio como función de pérdida aprende a llevar a cabo la baja de resolución.

5.9.3. Impacto en la arquitectura

La representación de la trayectoria como matriz binaria permitió emplear arquitecturas de GANs especializadas en el tratamiento de imágenes, como lo son las redes convolucionales. Se adaptó al generador (ver anexo A.4.4) para que emplee operaciones convolucionales en la generación de trayectorias, manteniendo las dimensiones de su entrada (vector de ruido), y redimensionando su salida para que tenga las dimensiones de una trayectoria de alta resolución. El discriminador también se adaptó para utilizar operaciones convolucionales.

La función que se utilizó para entrenar el módulo de downscaling debe contemplar la nueva representación de las trayectorias. Se cambió la función de downscaling por $T^L_{xy} = T^H_{(x+1)y} \vee T^H_{x(y+1)} \vee T^H_{(x-1)y} \vee T^H_{x(y-1)}$. Se volvió a entrenar el downscaler con la nueva función de downscaling, y se logró exitosamente obtener una red capaz de llevar a cabo una baja de resolución. La figura 5.18 muestra una comparación de una trayectoria, su downscaling mediante la función y la trayectoria que se obtuvo mediante la red una vez entrenada.



(a) Trayectoria de alta resolución

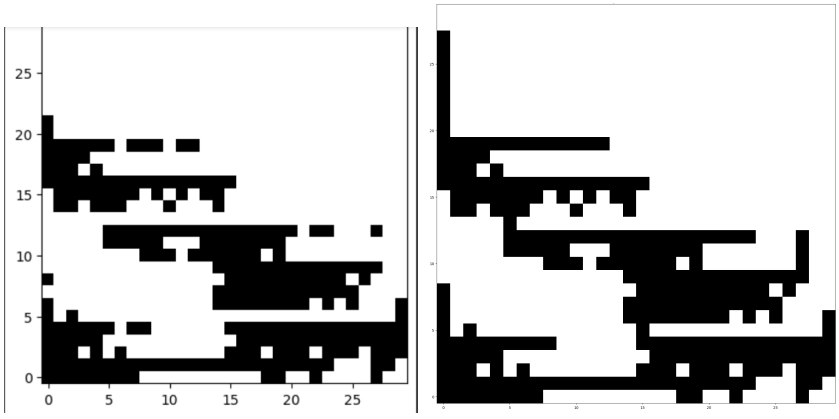
(b) Baja de resolución mediante la función de entrenamiento

(c) Baja de resolución mediante la red neuronal

Figura 5.18: Representación de desempeño de downscaler mediante red neuronal

Algunas de las trayectorias generadas con la codificación matricial no se ajustaron completamente a la lógica de vigilancia. Se observaron casos donde las trayectorias, representadas matricialmente, no tienen marcados como visitados puntos donde se encuentran los POIs, o estas trayectorias presentan puntos visitados disconexos del resto de los puntos visitados, por lo que sería imposible que se alcancen dichos puntos. Los resultados obtenidos reflejan que el 100 % de las trayectorias presentaron puntos aislados, con un promedio de 19.59 puntos aislados por trayectoria. Además, en el 27 % de las trayectorias se evidenció la falta de al menos un POI en las trayectorias generadas, aunque no se detectaron casos en los que ambos POIs estuvieran ausentes. Para solucionar estos errores en las trayectorias, se llevó a cabo un post-procesamiento de las trayectorias generadas por el generador, agregando puntos necesarios para hacer conexa a la trayectoria, y para incluir a todos los POIs dentro de la misma. Este post procesamiento solo se llevó a cabo sobre las imágenes generadas por el generador de la GAN una vez que se finalizó el entrenamiento. La figura 5.19 muestra el resultado del post procesamiento.

Al utilizar una representación de trayectorias mediante matriz binaria, el evaluador implementado no pudo utilizarse en la función de pérdida utilizada durante el entrenamiento, ya que, al no existir una dimensión temporal en la representación, la métrica que mide si los POIs son visitados cumpliendo con los tiempos requeridos, no puede ser evaluada. Las métricas que evalúan la factibilidad de la trayectoria, como la salida del área de vigilancia y los saltos dentro de la trayectoria, pierden relevancia, dado que esta representación elimina el



(a) Imagen generada por la GAN (b) Imagen corregida por el script

Figura 5.19: Antes y después del script de post procesamiento

problema de la salida del área y permite conectar fácilmente los puntos aislados a las componentes conexas. De esta manera, la única métrica que se evalúa es la métrica de cobertura, y se consideró que en este caso el evaluador no aportaría información pertinente al entrenamiento.

Con el fin de compensar la ausencia de la dimensión temporal en esta representación de las trayectorias se utilizó una variación de la heurística utilizada para generar los datos de entrenamiento. Esta heurística genera trayectorias de UAVs con la dimensión temporal, restringiendo el área de búsqueda a aquella generada por la GAN utilizando el algoritmo A* para el calculo de trayectorias entre dos puntos. El flujo de la solución final se encuentra representado en el diagrama 5.20

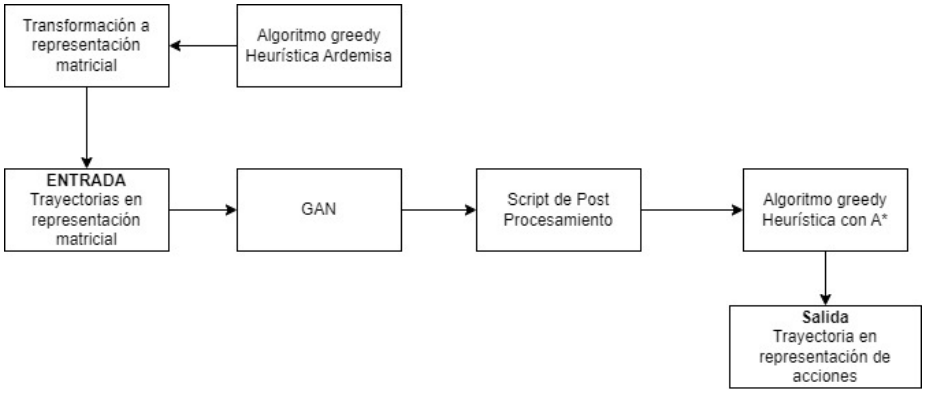


Figura 5.20: Flujo para la generación de trayectorias mediante GAN utilizando representación matricial

5.9.4. Resultados

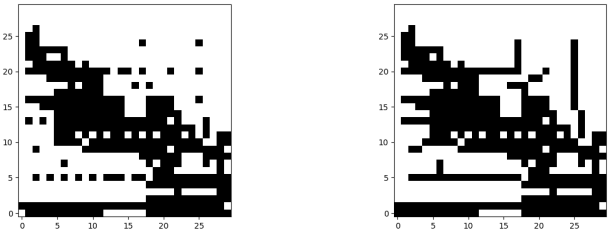
Se obtuvieron resultados donde el discriminador rápidamente superaba al generador y por ende este último no lograba aprender a generar trayectorias. Para solucionar este problema se disminuyó drásticamente el learning rate del discriminador para darle espacio al generador para poder aprender. Se realizaron varios experimentos con distintas configuraciones, algunas de estos experimentos se detallan en la tabla 5.4. Entre las distintos experimentos, se utilizó un Learning Reate Scheduler para hacer variar los learning rates del generador y del discriminador sin mayor éxito.

Cuadro 5.4: Resultados de la GAN con enfoque matricial

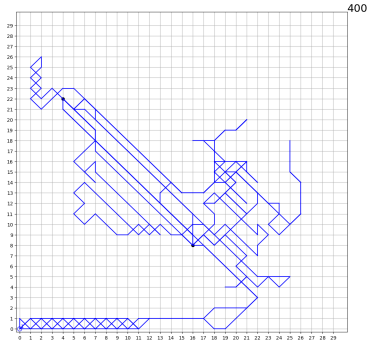
Nro de experimento	*GLR*	*DLR*	*D Loss*	*G Loss*
0	1×10^{-3}	1×10^{-5}	0.673	2.274
1	1×10^{-3}	1×10^{-6}	1.123	0.943
2	1×10^{-3}	3×10^{-6}	0.753	1.992

Los hiperparámetros del experimento uno brindaron los mejores resultados, evidenciado en la gráfica de la función de pérdida del generador y el discriminador a lo largo de las épocas de entrenamiento, y en las trayectorias generadas por la red.

La figura 5.22 muestra como ambas loses tiene una evolución equilibrada en donde ninguno de los dos componentes supera al otro. La figura 5.21 muestra la imagen generada por la GAN seguida de la imagen obtenida por el script de post procesamiento y por último la trayectoria obtenida por el algoritmo greedy con la heurística Yahera.



(a) Imagen generada por la GAN (b) Imagen corregida por el script



(c) Imagen que representa la trayectoria obtenida a partir del uso de la heurística con A*

Figura 5.21: Progreso del post procesamiento de la salida de la GAN

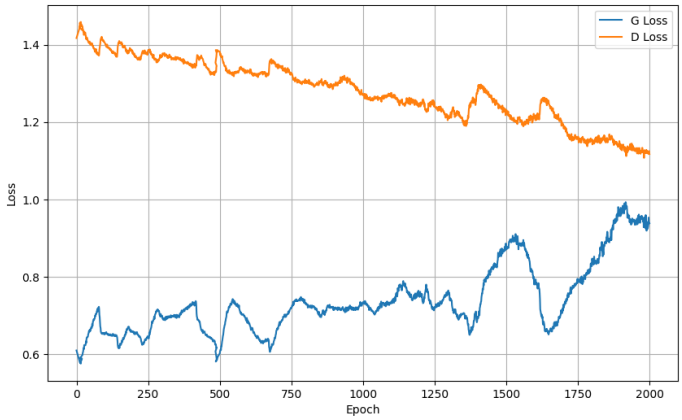


Figura 5.22: Gráfica de losses en función de las épocas para el experimento número uno

Capítulo 6

Conclusiones y Trabajo Futuro

En este capítulo se presentan conclusiones obtenidas a partir de lo estudiado, experimentado y detallado a lo largo del proyecto. También se presentan posibles lineas de trabajo futuro para profundizar la temática abordada.

6.1. Conclusiones

Se diseñó e implementó un algoritmo greedy que considera diversas características críticas del área de vigilancia, tales como la presencia de obstáculos, la ubicación de POIs que debían ser visitados, y la gestión de la batería de los UAVs. Este algoritmo fue clave para la creación de un set de datos de entrenamiento así como para la generación de trayectorias basadas en las imágenes producidas por la versión convolucional de la GAN. Las trayectorias generadas por el algoritmo greedy fueron inicialmente representadas en formato de acciones, lo que garantizó la continuidad de las trayectorias generadas por la GAN. Estas acciones fueron posteriormente recodificadas en otros formatos, como una representación en puntos cartesianos, la cual garantizó la generación de trayectorias comprendidas dentro del área, o una imagen a través de una matriz binaria, que permitió el uso de arquitecturas de GAN convolucionales. Esta versatilidad en las representaciones permitió un enfoque integral para el análisis y evaluación de las trayectorias en búsqueda de mejorar la calidad de los resultados obtenidos.

Se diseñó, implementó y entrenó una GAN para la generación de trayectorias a partir de los datos de entrenamiento. Esta red no hizo uso del módulo de downscaling y fue entrenada en el set de datos codificado en acciones generado por el algoritmo greedy. Esta implementación de la GAN no fue capaz de generar trayectorias utilizables, debido a que se llevaban a cabo fuera del área de vigilancia. El problema de la salida del área designada en las trayectorias se

manifestó en todas las codificaciones a las cuales la limitación del área no era inherente a la representación.

Se desarrolló una función de pérdida personalizada para el entrenamiento de la GAN, con el objetivo de dirigir al generador a producir trayectorias que se mantuvieran dentro del área de vigilancia. Este desarrollo incluyó la transición de TensorFlow a PyTorch y la implementación de un perceptrón multicapa. Inicialmente, se utilizó una combinación de BCE con una evaluación de trayectorias personalizada, pero los resultados no fueron satisfactorios. Para mejorar la efectividad, se implementó una estrategia de evolución de pesos, en la cual la ponderación entre BCE y la evaluación variaba de manera lineal y exponencial a lo largo de las épocas de entrenamiento. Sin embargo, se concluyó que estos enfoques dinámicos no solo no mejoraron el rendimiento del generador, sino que, en algunos casos, empeoraron el porcentaje de tiempo que el UAV permanecía dentro del área designada en comparación con el uso de pesos constantes. Estos resultados indican las dificultades inherentes a la personalización de funciones de pérdida en este tipo de redes y la necesidad de explorar otras estrategias más efectivas en futuros desarrollos.

Se desarrolló un módulo de downscaling que toma como entrada una trayectoria en alta resolución y la transforma en una de baja resolución. El propósito de este módulo es adaptar el generador para que produzca trayectorias de alta resolución, mientras que el discriminador se entrena con un dataset de baja resolución. Inicialmente, este módulo se implementó como una función que realiza transformaciones no lineales y retorna una nueva estructura, lo que resulta en la pérdida total de gradientes, detectada al observar que no había evolución en ninguna de las pérdidas durante el entrenamiento. Para solucionar este problema, se implementó el módulo de downscaling utilizando una red neuronal simple. Se realizaron varios intentos con la representación de acciones, pero estos no produjeron buenos resultados debido a que el cálculo de la pérdida entre dos acciones, basado en la diferencia cuadrática media, no reflejaba adecuadamente las similitudes entre diferentes acciones. Esta limitación llevó a la reestructuración de la representación de las trayectorias, optando finalmente por una representación cartesiana, con la que se obtuvieron resultados más efectivos. Este módulo se integró en la arquitectura entre la salida del generador y la entrada del discriminador, permitiendo así el entrenamiento de un generador de alta resolución.

Debido a la naturaleza secuencial del problema se desarrolló e implementó una LSTM GAN. Se obtuvieron mejoras respecto a la arquitectura de perceptrón multicapa de un 80 % en la distancia de saltos y un 20 % en la cantidad de saltos. Esta arquitectura si bien mejora los resultados obtenidos respecto a intentos previos aún no provee trayectorias viables para su uso.

Se experimentó con una arquitectura WGAN para la generación de trayectorias en codificación cartesiana. Mediante esta arquitectura se logró reducir la cantidad y la magnitud de los saltos presentes en las trayectorias en un 30 % y 89 % respectivamente. Estas mejoras evidencian el potencial de la arquitectura WGAN en la generación de sets de coordenadas cartesianas que requieran consistencia de localidad entre coordenadas adyacentes. A pesar de las mejoras

obtenidas, las trayectorias generadas por la WGAN presentaron el 50 % de los movimientos de los UAVs como saltos, por lo que mayor progreso con WGAN es necesario para que pueda ser utilizada para la generación de trayectorias cartesianas.

La culminación de este proyecto consiste en la generación de imágenes de trayectorias de alta resolución. Para llevar a cabo esta generación se tuvo que adaptar el modulo de downscaling y entrenarlo con la nueva representación matricial. Las trayectorias generadas en representación de imágenes carecen de la dimensión del tiempo. La interpretación de la imagen varia dependiendo el algoritmo de generación de trayectoria que se aplique, es por esto que la generación con datos tabulares tal como se propuso en el capitulo 4 es de suma importancia ya que se obtiene la totalidad de la información la cual no puede ser provista por una trayectoria representada a partir de una imagen. Para introducir la dimensión del tiempo se utilizó un script el cual a partir de la imagen genera con un algoritmo greedy, basado en el algoritmo greedy que se usó para crear el dataset, una secuencia de acciones que es la salida final de la experimentación.

6.2. Trabajo Futuro

A lo largo del desarrollo de este proyecto se han alcanzado importantes avances en la comprensión y aplicación de la generación de trayectorias de UAVs utilizando GANs. Sin embargo, como en toda investigación, han surgido nuevas preguntas y desafíos que abren la puerta a futuras exploraciones. En esta sección se identifica y discute las posibles direcciones que podrían tomarse para continuar y expandir el trabajo presentado.

Una de las líneas de trabajo futuro es la mejora del dataset de entrenamiento de las GANs, enfocándose en aumentar la variabilidad de las trayectorias generadas por el algoritmo greedy. En la fase actual del proyecto, las trayectorias fueron generadas utilizando un enfoque que permite la implementación de heurísticas personalizadas, siendo la heurística “Ardemisa” la empleada en este caso. Esta heurística permite que los UAVs realicen un movimiento exploratorio por el área, asignando la visita a un POI al primer UAV disponible en caso de que el POI así lo requiera. Adicionalmente, una vez transcurrido un tiempo determinado, el UAV regresa a la base para recargar su batería. Sin embargo, la variabilidad en estas trayectorias fue limitada principalmente a decisiones aleatorias durante el movimiento exploratorio. Una posible mejora identificada es la introducción de aleatoriedad en la selección del UAV encargado de visitar un POI, para evitar que siempre sea el mismo UAV el que asuma la responsabilidad, lo cual podría generar un conjunto de datos más diverso y representativo. Esta mayor diversidad en los datos de entrenamiento podría mejorar la capacidad de generalización de las GANs, permitiéndoles aprender de un conjunto de trayectorias más variado y, por ende, mejorar la calidad de las trayectorias generadas en aplicaciones futuras.

Durante el desarrollo del proyecto se encontraron diversos desafíos que hicieron necesario simplificar el escenario de estudio. Como resultado, se optó por limitar la experimentación a un solo UAV operando en un área específica. En este escenario simplificado, el UAV comienza y termina su trayectoria en el mismo punto de partida, sin considerar la recarga de la batería durante el vuelo y eliminando la presencia de obstáculos en su camino. Esta simplificación tuvo como objetivo principal obtener resultados iniciales que fueran prometedores con alguna de las arquitecturas empleadas. Sin embargo, se reconoce que este enfoque limitado no captura la complejidad de situaciones más realistas. Por lo tanto, se sugiere que futuras investigaciones exploren escenarios más completos, que incluyan múltiples UAVs operando de manera coordinada, la presencia de obstáculos que deben ser evitados durante la trayectoria, y la posibilidad de recargar la batería del UAV durante la misión de vigilancia. Estas extensiones permitirían evaluar de manera más precisa la eficacia de las arquitecturas en condiciones más cercanas a las aplicaciones reales.

En la experimentación realizada con la arquitectura WGAN, se constató un comportamiento interesante respecto al uso de optimizadores con momento, como Adam. El artículo original que presenta la arquitectura WGAN (Arjovsky y cols., 2017) menciona que el empleo de estos optimizadores puede generar inestabilidad en el entrenamiento de la red. Esta inestabilidad a causa de la utilización de optimizadores con momento se corroboró en las gráficas de las funciones de pérdida, donde se observaron fluctuaciones considerables. Sin embargo, a pesar de esta inestabilidad teórica y observable, los resultados obtenidos con Adam mostraron una notable mejora en la reducción de los saltos presentes en las trayectorias generadas, alcanzando una mejora aproximada del 45 %. Este resultado sugiere que, bajo ciertas condiciones, el uso de optimizadores con momento podría ser beneficioso para el entrenamiento de WGAN, incluso en el contexto de datos tabulares, donde tradicionalmente estas redes presentan desafíos adicionales. Dado que WGAN fue la arquitectura que produjo los mejores resultados en comparación con otras alternativas, se propone realizar una investigación más exhaustiva para entender mejor su comportamiento y optimización en escenarios similares, lo que podría abrir nuevas oportunidades para su aplicación en el procesamiento de datos tabulares.

Apéndice A

Anexo

A.1. Salida del algoritmo greedy

1 8 1 1 1 1 1 1 6 5 6 6 5 7 7 6 6 5 7 7 7 7 7 2 1 5 7 4 3 3 3 3 3 3 3 3 3 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 1 1
1 1 1 1 1 1 5 7 5 7 7 7 5 6 5 5 7 5 6 7 7 7 7 7 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 6 0 2 7 7 5 1 1 7 7 0 5 1 7 3 0 8 3 2 5 7 1 6
2 0 8 4 7 4 2 3 5 6 5 4 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 8 7 3 4 1 7 7 1 6 5 7 0 8 3 7 5 7 7 7 7 7 7 7 7 8 4 2 8 4 4 3
3 3 3 3 3 3 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 1 1 1 1 1 1 1 7 7 6 7 5 7 7
5 7 6 5 7 7 7 7 5 5 4 3 3 3 3 3 3 3 3 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 7 0 3 2 7 1 1 1 1 1 1 1 8 2 5 3 8 8 4 1 0
2 5 0 1 6 6 7 6 6 6 6 4 4 4 4 4 4 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

A.2. Custom Loss

```
from torch import ones
from torch.nn import BCELoss,Module
from env_parser import Env

class CustomLoss(Module):

    def __init__(self, evaluations, eval_weight, regular_weight):
        super(CustomLoss, self).__init__()
        self.evaluations = evaluations
        self.eval_weight = eval_weight
```

```
        self.regular_weight = regular_weight

    def adjust_weights(self,eval_weight:float,regular_weight:float):
        self.evalWeight = eval_weight
        self.regular_weight = regular_weight

    def set_evaluations(self,evaluations):
        self.evaluations = evaluations

    def forward(self, predictions, targets):
        env = Env.get_instance()
        loss_fun = BCELoss()
        reg = self.regular_weight * loss_fun(predictions, targets)
        evaluation = self.eval_weight * \
                     loss_fun(self.evaluations,ones(self.evaluations.size(0))
                     .to(env.DEVICE))
        total = reg + evaluation
        return total
```

A.3. Resultados del enfoque de acciones

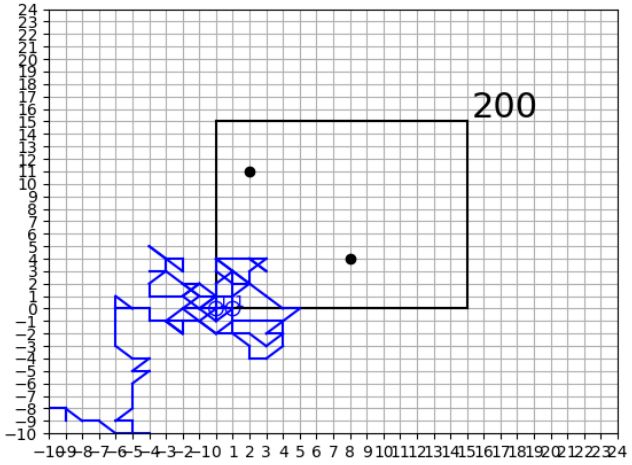


Figura A.1: Trayectoria generada por el generador sin la presencia del evaluador en la función de pérdida

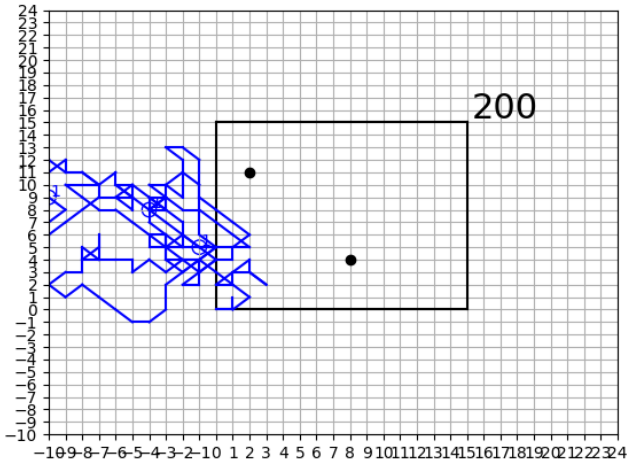


Figura A.2: Trayectoria generada por el generador con pesos adaptativos de forma lineal

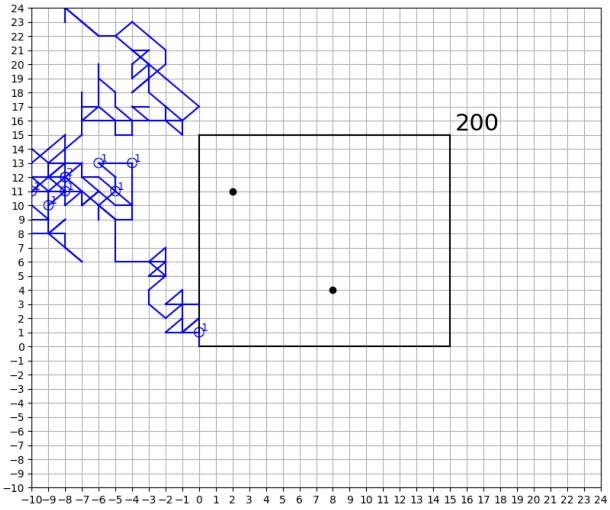


Figura A.3: Trayectoria generada por el generador con pesos adaptativos de forma exponencial

A.4. Arquitecturas de GAN utilizadas

A.4.1. Cartesian Perceptron

```
class Generator(Module):
    loss_fun: CustomLoss = CustomLoss(empty(1), 0, 0)
    def __init__(self):
        super(Generator, self).__init__()
        self.noise_dim = NOISE_DIMENSION
        self.main = Sequential(
            Linear(self.noise_dim, 256),
            LeakyReLU(0.2),
            Linear(256, 512),
            LeakyReLU(0.2),
            Linear(512, 1024),
            LeakyReLU(0.2),
            Linear(1024, UAV_AMOUNT * (HR_TOTAL_TIME * 2)),
            Sigmoid(),
        )
        self.optimizer = Adam(self.parameters(), lr=G_LEARN_RATE)
    def forward(self, x):
        out =  self.main(x).view(-1, UAV_AMOUNT, HR_TOTAL_TIME , 2)
        return out
    def custom_train(self, discriminator, data_fake, eval_tensor, epoch):
        curr_batch_size = data_fake.size(0)
        real_label = ones(curr_batch_size,1).to(DEVICE)
        self.optimizer.zero_grad()
        output = discriminator(data_fake)
        eval_weight, regular_weight = WeightApproach.get_instance().get_weights(epoch)
        self.loss_fun.adjust_weights(eval_weight, regular_weight)
        self.loss_fun.set_evaluations(eval_tensor)
        loss = self.loss_fun(output, real_label)
        loss.backward()
        self.optimizer.step()
        return loss

class Discriminator(Module):
    loss_function: _Loss = BCELoss()
    def __init__(self):
        super(Discriminator, self).__init__()
        self.n_input = UAV_AMOUNT * TOTAL_TIME *2
        self.main = self._build_model()
        self.optimizer = Adam(self.parameters(), lr=D_LEARN_RATE)
    def _build_model(self):
        model = Sequential(
            Linear(self.n_input, 1024),
```

```
            LeakyReLU(0.2),
            Dropout(0.4),
            Linear(1024, 512),
            LeakyReLU(0.2),
            Dropout(0.4),
            Linear(512, 256),
            LeakyReLU(0.2),
            Dropout(0.4),
            Linear(256, 1),
            Sigmoid(),
        )
        self._initialize_weights(model)
        return model
    def _initialize_weights(self, model):
        for m in model.modules():
            if isinstance(m, Linear):
                kaiming_normal_(m.weight.data)  # Inicialización de He
                if m.bias is not None:
                    constant_(m.bias.data, 0)
        return model
    def forward(self, x):
        x = x + randn(x.size()).to(DEVICE) * 0.1
        x = x.view(-1, self.n_input)
        return self.main(x)
    def custom_train(self, data_real, data_fake: Tensor):
        curr_batch_size = data_real.size(0)
        real_label = ones(curr_batch_size,1).to(DEVICE)
        fake_label = zeros(curr_batch_size).to(DEVICE)
        self.optimizer.zero_grad()
        output_real = self(data_real)
        real_smooth_label = self.label_smoothing(real_label)
        loss_real = self.loss_function(output_real, real_smooth_label)
        output_fake = self(data_fake)
        fake_smooth_label = self.label_smoothing(fake_label).unsqueeze(1)
        loss_fake = self.loss_function(output_fake, fake_smooth_label)
        loss_real.backward()
        loss_fake.backward()
        self.optimizer.step()
        return loss_real + loss_fake
    def label_smoothing(self, target, smoothing=0.2):
        return target * (1 - smoothing) + 0.5 * smoothing
```

A.4.2. Cartesian WGAN

```
class Generator(Module):
    def __init__(self):
```

```
        super(Generator, self).__init__()
        self.noise_dim = NOISE_DIMENSION
        self.main = self._build_model()
        self.optimizer = SGD(self.parameters(), lr=G_LEARN_RATE, momentum=0.9)
    def _build_model(self):
        model = Sequential(
            Linear(self.noise_dim, 256),
            LeakyReLU(0.2),
            Linear(256, 512),
            LeakyReLU(0.2),
            Linear(512, 1024),
            LeakyReLU(0.2),
            Linear(1024, UAV_AMOUNT * (HR_TOTAL_TIME * 2)),
            Sigmoid()
        )
        return model
    def forward(self, x):
        return self.main(x).view(-1, UAV_AMOUNT, HR_TOTAL_TIME, 2)
    def custom_train(self, discriminator, hr_fake_data, eval_tensor, epoch):
        self.optimizer.zero_grad()
        fake_data = downscaler(hr_fake_data)
        loss = -discriminator(fake_data).mean()
        loss.backward()
        self.optimizer.step()
        return loss.item()

class Discriminator(Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.n_input = UAV_AMOUNT * TOTAL_TIME * 2
        self.main = self._build_model()
        self.optimizer = SGD(self.parameters(), lr=D_LEARN_RATE, momentum=0.9)
    def _build_model(self):
        model = Sequential(
            Linear(self.n_input, 1024),
            LeakyReLU(0.2),
            Dropout(0.2),
            Linear(1024, 512),
            LeakyReLU(0.2),
            Dropout(0.2),
            Linear(512, 256),
            LeakyReLU(0.2),
            Dropout(0.2),
            Linear(256, 1),
        )
        self._initialize_weights(model)
```

```
        return model
    def _initialize_weights(self, model):
        for m in model.modules():
            if isinstance(m, Linear):
                init.kaiming_normal_(m.weight.data)  # Inicialización de He
                if m.bias is not None:
                    init.constant_(m.bias.data, 0)
        return model
    def forward(self, x):
        x = x + randn(x.size()).to(DEVICE) * 0.1
        x = x.view(-1, self.n_input)
        return self.main(x)
    def custom_train(self, real_data, fake_data):
        self.optimizer.zero_grad()
        real_loss = self(real_data).mean()
        fake_loss = self(fake_data).mean()
        loss = fake_loss - real_loss
        loss.backward()
        self.optimizer.step()
        for p in self.parameters():
            p.data.clamp_(-0.01, 0.01)
        return loss.item()
```

A.4.3. Cartesian LSTM GAN

```
class Generator(Module):
    optimizer: Optimizer
    loss_fun: CustomLoss = CustomLoss(empty(1), 0, 0)

    def __init__(self):
        super(Generator, self).__init__()
        self.noise_dim = NOISE_DIMENSION
        self.linear_1 = Linear(self.noise_dim, 256)
        self.linear_2 = Linear(256, 512)
        self.linear_4 = Linear(1024, UAV_AMOUNT * (HR_TOTAL_TIME * 2))
        self.lstm = LSTM(512, 1024,1, batch_first=True)
        self.sigmoid = Sigmoid()
        self.leakyRelu = LeakyReLU(0.2)
        self.optimizer = Adam(self.parameters(), lr=G_LEARN_RATE)

    def forward(self, x):
        x = self.linear_1(x)
        x = self.leakyRelu(x)
        x = self.linear_2(x)
        x = self.leakyRelu(x)
        x, _ = self.lstm(x)
```

```
        x = self.leakyRelu(x)
        x = self.linear_4(x)
        x = self.leakyRelu(x)
        x = self.sigmoid(x)
        out = x.view(-1, UAV_AMOUNT, HR_TOTAL_TIME , 2)
        return out

    def custom_train(self, discriminator, data_fake, eval_tensor, epoch):
        curr_batch_size = data_fake.size(0)
        real_label = ones(curr_batch_size,1).to(DEVICE)
        self.optimizer.zero_grad()
        output = discriminator(data_fake)
        eval_weight, regular_weight = WeightApproach.get_instance().get_weights(epoch)
        self.loss_fun.adjust_weights(eval_weight, regular_weight)
        self.loss_fun.set_evaluations(eval_tensor)
        loss = self.loss_fun(output, real_label)
        loss.backward()
        self.optimizer.step()
        return loss

class Discriminator(Module):
    loss_function: _Loss = BCELoss()
    optimizer: Optimizer

    def __init__(self):
        super(Discriminator, self).__init__()
        self.n_input = UAV_AMOUNT * TOTAL_TIME *2
        self.main = self._build_model()
        self.optimizer = Adam(self.parameters(), lr=D_LEARN_RATE)

    def _build_model(self):
        model = Sequential(
            Linear(self.n_input, 1024),
            LeakyReLU(0.2),
            Dropout(0.4),
            Linear(1024, 512),
            LeakyReLU(0.2),
            Dropout(0.4),
            Linear(512, 256),
            LeakyReLU(0.2),
            Dropout(0.4),
            Linear(256, 1),
            Sigmoid(),
        )
        self._initialize_weights(model)
        return model
```

```
    def _initialize_weights(self, model):
        for m in model.modules():
            if isinstance(m, Linear):
                kaiming_normal_(m.weight.data)  # Inicialización de He
                if m.bias is not None:
                    constant_(m.bias.data, 0)
        return model

    def forward(self, x):
        x = x + randn(x.size()).to(DEVICE) * 0.1
        x = x.view(-1, self.n_input)
        return self.main(x)

    def custom_train(self, data_real, data_fake: Tensor):
        curr_batch_size = data_real.size(0)
        real_label = ones(curr_batch_size,1).to(DEVICE)
        fake_label = zeros(curr_batch_size).to(DEVICE)
        self.optimizer.zero_grad()
        output_real = self(data_real)
        real_smooth_label = self.label_smoothing(real_label)
        loss_real = self.loss_function(output_real, real_smooth_label)
        output_fake = self(data_fake)
        fake_smooth_label = self.label_smoothing(fake_label).unsqueeze(1)
        loss_fake = self.loss_function(output_fake, fake_smooth_label)
        loss_real.backward()
        loss_fake.backward()
        self.optimizer.step()
        return loss_real + loss_fake

    def label_smoothing(self, target, smoothing=0.2):
        return target * (1 - smoothing) + 0.5 * smoothing
```

A.4.4. Image Convolutional GAN

```
class Generator(Module):
    loss_fun: CustomLoss = CustomLoss(empty(1), 0, 0)
    def __init__(self):
        super(Generator, self).__init__()
        self.noise_dim = env.NOISE_DIMENSION
        output_dim_x, output_dim_y = env.ENVIRONMENT_X_AXIS, env.ENVIRONMENT_Y_AXIS
        self.main = Sequential(
            ConvTranspose2d(self.noise_dim, 512, 2, 1, 0),
            LeakyReLU(0.2),
            ConvTranspose2d(512, 256, 4, 2, 1),
            LeakyReLU(0.2),
```

```
            ConvTranspose2d(256, 128, 4, 2, 1),
            LeakyReLU(0.2),
            ConvTranspose2d(128, 64, 4, 2, 1),
            LeakyReLU(0.2),
            ConvTranspose2d(64, 1, 4, 2, 1),
            Sigmoid()
        )
        dummy_input = randn(1,self.noise_dim)
        dummy_input = dummy_input.view(-1,self.noise_dim,1,1)
        for layer in self.main:
          dummy_input = layer(dummy_input)
          print(f"Output shape after {layer.__class__.__name__}: {dummy_input.shape}")
    def forward(self, x):
        x = x.view(-1, self.noise_dim, 1, 1)
        out = self.main(x)
        out = interpolate(out, size=(30, 30), mode='nearest')
        return out
    def set_optimizer(self, optimizer: Optimizer):
        self.optimizer = optimizer
    def custom_train(
        self, discriminator: Discriminator, data_fake, eval_tensor, epoch: int
    ):
        curr_batch_size = data_fake.size(0)
        real_label = label_real(curr_batch_size)
        self.optimizer.zero_grad()
        output = discriminator(data_fake)
        eval_weight, regular_weight = WeightApproach.get_instance().get_weights(epoch)
        self.loss_fun.adjust_weights(eval_weight, regular_weight)
        self.loss_fun.set_evaluations(eval_tensor)
        loss = self.loss_fun(output, real_label)
        loss.backward()
        self.optimizer.step()
        return loss

class Discriminator(Module):
    loss_function: _Loss = BCELoss()
    def __init__(self):
      super(Discriminator, self).__init__()
      self.main = Sequential(
          Conv2d(1, 64, kernel_size=4, stride=2, padding=1),
          LeakyReLU(0.2),
          Dropout(0.4),
          Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
          LeakyReLU(0.2),
          Dropout(0.4),
          Flatten(),
```

```
      Linear(128 * (15 // 4) * (15 // 4), 1),
      Sigmoid()
  )
  self.optimizer = Adam(self.parameters(), lr=env.D_LEARN_RATE)
def set_optimizer(self, optimizer: Optimizer):
    self.optimizer = optimizer
def _initialize_weights(self, model):
    for m in model.modules():
        if isinstance(m, Linear):
            init.kaiming_normal_(m.weight.data)  # Inicialización de He
            if m.bias is not None:
                init.constant_(m.bias.data, 0)
    return model
def forward(self, x):
    x = x + randn(x.size()).to(env.DEVICE) * 0.2
    return self.main(x)


def custom_train(self, data_real: Tensor, data_fake: Tensor):
    curr_batch_size = data_real.size(0)
    real_label = label_real(curr_batch_size)
    fake_label = label_fake(curr_batch_size)
    self.optimizer.zero_grad()
    output_real = self(data_real)
    real_smooth_label = self.label_smoothing(real_label)
    loss_real = self.loss_function(output_real, real_smooth_label)
    output_fake = self(data_fake)
    fake_smooth_label = self.label_smoothing(fake_label)
    loss_fake = self.loss_function(output_fake, fake_smooth_label)
    loss_real.backward()
    loss_fake.backward()
    self.optimizer.step()
    return loss_real + loss_fake

def label_smoothing(self, target, smoothing=0.2):
    return target * (1 - smoothing) + 0.5 * smoothing
```

Referencias

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems [Manual de software informático]. Descargado de `https://www.tensorflow.org/` (Version 2.5.0, Retrieved July 9, 2024)

Acero, A., Huang, Y., y Jiang, Y. (2023, 04). A review of unmanned aerial vehicle applications in construction management: 2016–2021. *Standards*, *3*, 95-109. doi: 10.3390/standards3020009

Arjovsky, M., Chintala, S., y Bottou, L. (2017). *Wasserstein gan.*

Bahdanau, D., Cho, K., y Bengio, Y. (2016). *Neural machine translation by jointly learning to align and translate.*

Behak, S., Rondán, G., Zanetti, M., Iturriaga, S., y Nesmachnow, S. (2020). Distributed greedy approach for autonomous surveillance using unmanned aerial vehicles..

Cuerno, C., Garcia, L., Sanchez, A., Carrió, A., Sanchez, J. L., y Campoy, P. (2016). Evolución histórica de los vehículos aéreos no tripulados hasta la actualidad. *Dyna*, *91*(3), 282–288.

Dechter, R., y Pearl, J. (1985, jul). Generalized best-first search strategies and the optimality of a*. *J. ACM*, *32*(3), 505–536. Descargado de `https://doi.org/10.1145/3828.3830` doi: 10.1145/3828.3830

Gaudín, A., Madruga, G., Rodríguez, C., Iturriaga, S., Nesmachnow, S., Paz, C., ... Bouvry, P. (2019). Autonomous flight of unmanned aerial vehicles using evolutionary algorithms..

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning.* MIT Press. (`http://www.deeplearningbook.org`)

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). *Generative adversarial networks.*

Gordon, C. (2024, 12 de marzo). Chatgpt and generative ai innovations are creating sustainability havoc. *Forbes*. Descargado de `https://www.forbes.com/sites/cindygordon/2024/03/12/chatgpt-and-generative-ai-innovations-are-creating-sustainability-havoc/`

Hochreiter, S., y Schmidhuber, J. (1997, 12). Long short-term memory. *Neural computation*, *9*, 1735-80. doi: 10.1162/neco.1997.9.8.1735

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in science & engineering*, *9*(3), 90–95.

Liu, X., Chen, H., y Andris, C. (2018). Trajgans: Using generative adversarial networks for geo-privacy protection of trajectory data. University Park, PA: The Pennsylvania State University. Descargado de https://ptal-io.github.io/lopas2018/papers/LoPaS2018_Liu.pdf

Mansouri, S. S., Karvelis, P., Georgoulas, G., y Nikolakopoulos, G. (2017). Remaining useful battery life prediction for uavs based on machine learning. *IFAC-PapersOnLine*, *50*(1), 4727–4732. (Konferensartikel i tidskrift) doi: 10.1016/j.ifacol.2017.08.863

MasterClass. (2023). *Zero-sum game meaning: Examples of zero-sum games.* Descargado de https://www.masterclass.com/articles/zero-sum-game-meaning#what-is-a-zerosum-game (2023-07-08)

Mohammadi, M., Al-Fuqaha, A., y Oh, J.-S. (2018, 07). Path planning in support of smart mobility applications using generative adversarial networks. En (p. 878-885). doi: 10.1109/Cybermatics_2018.2018.00168

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). *Pytorch: An imperative style, high-performance deep learning library.* Descargado de https://pytorch.org/ (Version 1.9.0, Retrieved July 9, 2024)

Patel, A. (2024). *Introduction to a*.* Descargado de https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html (Accedido en 2024)

Pombo, S. (2023). *Publicación sobre drones profesionales y desarrollo territorial.* Descargado de https://www.linkedin.com/posts/sebapombo_dronesprofesionales-desarrolloterritorial-activity-7083543657038909440-bp6A/?utm_source=share&utm_medium=member_desktop (Accedido: 2023-07-10)

Puente-Castro, A., Rivero, D., Pazos, A., y Fernandez-Blanco, E. (2022, 09). Uav swarm path planning with reinforcement learning for field prospecting. *Applied Intelligence*, *52*(12), 14101-14118. Descargado de https://doi.org/10.1007/s10489-022-03254-4 doi: 10.1007/s10489-022-03254-4

Python Software Foundation. (1991). Python programming language [Manual de software informático]. https://www.python.org/. (Version 3.10, Retrieved July 9, 2024)

Rao, J., Gao, S., Kang, Y., y Huang, Q. (2020). LSTM-TrajGAN: A Deep Learning Approach to Trajectory Privacy Protection. En K. Janowicz y J. A. Verstegen (Eds.), *11th international conference on geographic information science (giscience 2021) - part i* (Vol. 177, pp. 12:1–12:17). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. Descargado de https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.GIScience.2021.I.12 doi: 10.4230/LIPIcs.GIScience.2021.I.12

Saatchi, Y., y Wilson, A. G. (2017). *Bayesian gan.* Descargado de https://arxiv.org/abs/1705.09558

Sousa, M. (2019). *Visualizing the fundamentals of convolutional neural networks.* Descargado de https://towardsdatascience.com/

`visualizing-the-fundamentals-of-convolutional-neural-networks-6021e5b07f69` (2023-07-08)

Watson, A. (2020). *Deep learning techniques for super-resolution in video games.*