Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AGRONÓMICA Y BIOCIENCIAS

NEKAZARITZAKO INGENIARITZAKO ETA BIOZIENTZIETAKO GOI MAILAKO ESKOLA TEKNIKOA

Aprendizaje por refuerzo para control activo de flujo en túneles de viento con múltiples ventiladores

presentado por

Iñaki Elcano Esparza

aurkeztua

GRADO EN CIENCIA DE DATOS
GRADUA DATUEN ZIENTZIAN

Mayo, 2025 / *2025, maiatza*

Resumen

En este trabajo, se ha desarrollado un sistema de control inteligente para la gestión dinámica del flujo de aire en un túnel de viento de múltiples ventiladores(FAWT). El objetivo principal ha sido modelar la propagación de la velocidad del aire en el túnel mediante técnicas de aprendizaje automático, aprendizaje por refuerzo y convoluciones, permitiendo predecir y ajustar la distribución del flujo generado. De esta manera, se ha implementado un modelo de difusión estacionaria con convoluciones 3×3 para simular la propagación del flujo, generando un dataset que relaciona las velocidades iniciales con las resultantes aguas abajo. Se entrenó una red neuronal optimizando hiperparámetros como el *learning rate*, la regularización L2 y distintos optimizadores, asegurando la conservación de la energía cinética. Además, se ha conseguido en el conjunto test un error cuadrático medio de 0.00295 entre las velocidades deseadas y las generadas por el modelo.

Palabras clave: Redes neuronales, aprendizaje automático, filtros, aprendizaje supervisado.

Abstract

In this work, an intelligent control system was developed for the dynamic management of airflow in a multi-fan wind tunnel (FAWT). The main objective was to model air velocity propagation in the tunnel using machine learning, reinforcement learning, and convolution techniques, allowing the prediction and adjustment of the generated flow distribution. A stationary diffusion model with 3x3 convolutions was implemented to simulate flow propagation, generating a dataset that relates the initial velocities with the resulting downstream velocities. A neural network was trained by optimizing hyperparameters such as the learning rate, L2 regularization, and various optimizers, ensuring the conservation of kinetic energy. Furthermore, a root mean square error of 0.00295 was achieved in the test set between the desired velocities and those generated by the model.

Keywords: Neural networks, machine learning, filters, supervised learning.

Índice

1. Introducción

Un túnel de viento es una herramienta de investigación en la que se genera un flujo de aire controlado mediante ventiladores o aire comprimido, se desarrolla para ayudar en el estudio de los efectos del movimiento del aire alrededor de objetos sólidos[1]. Dentro del túnel de viento, el modelo permanece estacionario mientras se simulan las condiciones atmosféricas que experimentará el objeto de la investigación en una situación real. De esta forma, nos permite analizar diferentes procesos aerodinámicos, como distribuciones de presiones o fuerzas de arrastre sobre diferentes objetos, como aviones, edificios o puentes.

Debido a esto, surge la necesidad de hacer modelos matemáticos que nos permitan predecir y controlar el comportamiento del flujo de aire en el túnel de viento. Los modelos basados en las ecuaciones de Navier-Stokes completas son muy costosos computacionalmente y excesivamente complejos de resolver en algunos casos.

Cabe resaltar que existen dos tipos de túneles de viento: los convencionales y los digitales[2]. Los túneles convencionales únicamente tienen un único ventilador que generará todo el flujo de aire. Debido a esto, solo puede variar el flujo de manera uniforme en el túnel de viento. Por otro lado, tenemos los túneles digitales que están compuestos por una matriz de ventiladores, por lo tanto, se puede controlar y manejar el flujo de manera más precisa. En este trabajo modelaremos el flujo de un túnel de viento digital.

Por ello, en este trabajo se propone un enfoque simplificado, en el que se modeliza el túnel de viento únicamente a través de un mecanismo de difusión estacionaria de velocidades. Este modelo tiene como objetivo principal facilitar el control inverso del flujo: a partir de un estado de velocidades deseado en la salida del túnel, calcular las condiciones iniciales que los ventiladores deben imponer en la entrada para lograr dicho estado final.

Se han entrenado diferentes modelos utilizando diferentes técnicas del conocimiento, por un lado, se han aplicado técnicas basadas en el aprendizaje por refuerzo y por otro lado, redes neuronales. Se ha comenzado aplicando Q_learning y DQN, viendo la efectividad de estas técnicas y los resultados obtenidos. Posteriormente, se ha entrenado un modelo aplicando redes neuronales, que será con el que hagamos la simulación final del túnel de viento.

Para abordar este problema, vamos a construir un problema físico estacionario de difusión, en el que vamos a simular cómo las velocidades se propagan a través del túnel de viento. Sobre este modelo crearemos unos datos sintéticos que relacionarán las entradas de velocidades con las salidas de velocidades. Luego entrenaremos el modelo inverso que habrá aprendido a inferir las velocidades iniciales correctas para unas velocidades al final del túnel de viento deseadas.

1.1. Objetivos

El objetivo principal es modelizar el comportamiento del túnel de viento para poder controlar su flujo a lo largo del túnel de manera precisa. En concreto, se busca determinar qué condiciones de velocidad inicial deben imponerse en la entrada del túnel (control de ventiladores) para conseguir al final del túnel unas velocidades deseadas en los 24 puntos de medición distribuidos en la sección de salida.

De esta manera, el modelo permitirá no solo simular la evolución del flujo, sino también invertir el problema a partir de una configuración objetivo en la salida, calcular cuáles deben ser las velocidades iniciales que los ventiladores deben generar. Esta capacidad de control es fundamental para ensayos de aerodinámica, simulación de flujos personalizados y optimización de experimentos en túneles de viento.

Cabe distinguir la existencia de dos tipos de túneles de viento: el convencional y el digital. El túnel de viento convencional consiste en un solo ventilador, el que generará el flujo de aire de este modo, solo se puede variar la velocidad de manera uniforme en todo el túnel de viento. Por otro lado, tenemos el túnel de viento digital, que consiste en una matriz de ventiladores, por lo que se puede manejar de manera más precisa el control de flujo, nosotros modelaremos un túnel de este tipo.

Para llevar a cabo este control del flujo, crearemos un modelo físico estacionario basado en la difusión de las velocidades a través del túnel de viento.

Por otro lado, para entrenar este modelo, crearemos un conjunto de datos sintéticos que relacionará las velocidades iniciales con las velocidades finales.

Además, haremos un estudio exhaustivo de los principales factores que afectan al rendimiento del modelo, como por ejemplo, el efecto del tamaño del conjunto de datos de entrenamiento , la influencia de la estructura de la red neuronal (número de capas, número de neuronas por capa), el impacto de los hiperparámetros de entrenamiento (función de activación, optimizador, tamaño de batch, número de épocas). La sensibilidad del modelo al tamaño de la malla de simulación.

Finalmente, se pretende analizar los resultados obtenidos mediante gráficos de varias formas, como por ejemplo ondas, triángulos, círculos o cuadrados, evaluando cuantitativamente la calidad de las soluciones mediante métricas de error como el MSE (Error Cuadrático Medio).

1.1.1. Conocimiento previo

En este caso, en el edificio Jerónimo de Ayanz del campus de Arrosadía disponemos de un túnel de viento compuesto de 4 módulos y, a su vez, cada módulo tiene 3×2 ventiladores, es decir, 3 filas de ventiladores por 2 columnas de ventiladores; por tanto, el total del túnel de viento está compuesto por 4 filas de ventiladores por 6 columnas, por lo que hace un total de 24 ventiladores.

Por consiguiente, tras un estudio previo, se dio a conocer que la propagación depende del tipo de ventilador:

Para aquellos ventiladores situados en posiciones interiores de la malla (es decir, que no se encuentren ni en los bordes ni en las esquinas), la velocidad final en la posición (i, j) se calcula como una combinación ponderada de las velocidades de los ventiladores vecinos:

- 40 % de la velocidad del ventilador situado en (i, j).
- 10 % de la velocidad del ventilador superior $(i - 1, j)$.
- 10 % de la velocidad del ventilador inferior $(i + 1, j)$.
- 10 % de la velocidad del ventilador a la izquierda $(i, j - 1)$.
- 10 % de la velocidad del ventilador a la derecha $(i, j + 1)$.
- 5 % de la velocidad de cada uno de los ventiladores en posiciones diagonales:
  - Superior izquierda $(i - 1, j - 1)$,
  - Superior derecha $(i - 1, j + 1)$,
  - Inferior izquierda $(i + 1, j - 1)$,
  - Inferior derecha $(i + 1, j + 1)$.

En los laterales (bordes pero no esquinas)

Para ventiladores situados en los bordes de la malla pero no en las esquinas, se ajustan los pesos de la siguiente forma:

- 60 % de la velocidad del ventilador (i, j).
- 10 % de la velocidad del ventilador superior $(i - 1, j)$ (si existe).

- 10 % de la velocidad del ventilador inferior $(i+1, j)$ (si existe).
- 10 % de la velocidad del ventilador adyacente lateral (izquierda $(i, j-1)$ o derecha $(i, j+1)$, dependiendo de la posición).
- 5 % de la velocidad de cada uno de los ventiladores en las posiciones diagonales accesibles.

Se deben tener en cuenta las posiciones relativas para manejar correctamente las celdas que carecen de algunos vecinos debido a estar situadas en los bordes.

Puntos en las esquinas

Finalmente, para ventiladores situados en las esquinas de la malla, el reparto de velocidades se define como:

- 60 % de la velocidad del propio ventilador (i, j).
- 15 % de la velocidad de cada ventilador adyacente en dirección horizontal y vertical (por ejemplo, el ventilador de la derecha y el de abajo para la esquina superior izquierda).
- 10 % de la velocidad del ventilador en la posición diagonal correspondiente.

Este esquema de pesos simplificado permite aproximar la propagación de la velocidad dentro del túnel de viento de una manera física y computacionalmente eficiente, garantizando la conservación de la energía en el sistema.

A continuación en la **Figura 1** vemos de forma visual la propagación de los ventiladores, dependiendo de su tipo:

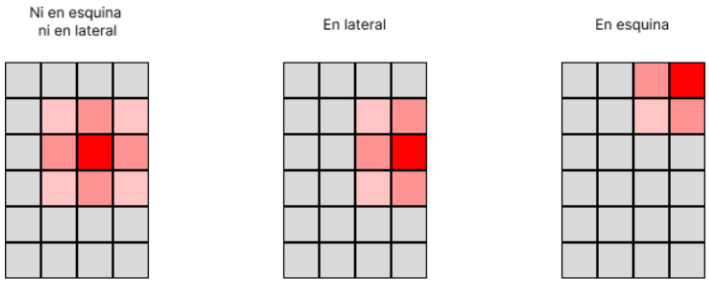


Figura 1: Propagación del flujo según tipo de ventilador.

2. Estado del arte

Modelado de túneles de viento

Como hemos comentado anteriormente, los túneles de viento son una herramienta fundamental para el estudio experimental de flujos de aire en cualquier rama de la investigación. El objetivo

El modelado se basa en la resolución de las ecuaciones de Navier-Stokes[3], que describen el comportamiento de los fluidos. No obstante, debido a la complejidad computacional y su difícil resolución, muchas veces se toman simplificaciones y aproximaciones para llevar a cabo determinados modelos.

Ecuaciones de Navier-Stokes:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\, \mathbf{u}) = 0, \tag{1}$$

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u}\right) = -\nabla p + \mu\, \nabla^2 \mathbf{u} + \rho\, \mathbf{f}. \tag{2}$$

Modelado Simplificado de difusión

Una de las simplificaciones que se pueden hacer en el modelado de túneles de viento es la difusión pura, es decir, que no se tiene en cuenta otras características como la presión o

el trasporte advectivo. De este modo, con la difusión pura [4] se verá cómo las velocidades se suavizan espacialmente a medida que avanza el flujo. Además, se va a aplicar un modelo estacionario, por lo que las condiciones del sistema no cambian con el tiempo.

Aplicaciones de redes neuronales a problemas inversos

En los problemas inversos, empezamos observando los efectos que diferentes simulaciones o parámetros afectan en nuestro modelo. Para estos problemas son de gran utilidad las redes neuronales arquitecturas como autodecoders[5], que han abordado tareas inversas de imagen aprendiendo del mapeo inverso. Debido a que son capaces de aprender relaciones no lineales, ofrecen estimaciones muy precisas en una sola ejecución una vez ya entrenada la red neuronal.

Redes Neuronales Inversa en Dinámica de fluidos

En la dinámica de fluidos, las aplicaciones de aprendizaje automático se han centrado en predicciones directas(predecir la evolución de un flujo), recientemente han comenzado a llevarse a cabo trabajos con redes neuronales para abordar problemas inversos.

Entre estos problemas destacan aquellos que buscan inferir las condiciones de frontera o patrones de inflado óptimos para lograr objetivos en el flujo final. Por ejemplo, Cheng y Zhang[6]implementaron un Res-PINN en el que se infieren directamente parámetros de contorno como la viscosidad mediante observaciones del campo de velocidades, consiguiendo muy buenos resultados.

Motivación del enfoque propuesto

Los métodos tradicionales necesitarían una gran cantidad de simulaciones para poder abordar este problema. Se ha desarrollado el enfoque comentado anteriormente que consiste en el entrenamiento con redes neuronales del problema inverso que aprenderá a predecir las velocidades iniciales a partir del estado final deseado. De esta manera, reduciremos el coste computacional y aumentaremos la flexibilidad para adaptarse a diferentes formas.

3. Aprendizaje por refuerzo

3.1. Q-learning

Una de las técnicas más populares dentro del aprendizaje por refuerzo es el **Q-learning**. Q-learning se basa en la estimación de los denominados **valores Q** ($Q(s, a)$)[7], que representan la utilidad esperada de realizar una acción a en un estado s, y luego seguir una política óptima.

El aprendizaje Q permite al agente utilizar las recompensas del entorno para aprender, con el tiempo, la mejor acción a tomar en un estado determinado. El agente aprenderá de la tabla Q. El agente de Q-learning aprende a comportarse adecuadamente a través de un proceso de prueba y error, donde va explorando las consecuencias de sus acciones. A medida que el agente se enfrenta a diferentes situaciones (llamadas estados) y toma decisiones (acciones), va recibiendo recompensas que le indican qué tan buena fue su elección.

3.1.1. Conceptos

-El ventilador sería en este caso nuestro agente, cuyo objetivo es ajustar su velocidad para alcanzar condiciones óptimas de flujo en el túnel de viento.

- Los estados representan las diferentes situaciones posibles que el ventilador puede enfrentar. En este caso, un estado puede corresponder a una determinada velocidad del flujo de aire, expresada en kilómetros por hora (km/h), que el sistema desea alcanzar o mantener.

-Nuestros agentes reaccionan realizando una acción para pasar de un estado a otro, en este caso, podemos aumentar, disminuir o mantener la velocidad en x kilómetros por hora.

-Después de una acción y cambiar de estado, el agente recibe una recompensa, esta recompensa puede ser positiva si las velocidades se acercan al estado óptimo o negativas o nulas si se alejan de las velocidades óptimas deseadas.

-La política es la estrategia de elegir una acción dado un estado con la esperanza de obtener mejores resultados.

Exploración frente a recompensas inmediatas

Una de las consideraciones clave es que buscar recompensas inmediatas no siempre conduce a las mejores soluciones a largo plazo. Adoptar una estrategia codiciosa, en la

que el agente siempre selecciona la acción que ofrece la mayor recompensa instantánea, esto puede provocar caer en subóptimos locales y quedarse estancado. Para ello, hay que equilibrar la exploración de nuevas soluciones con la explotación (elegir la mejor acción conocida hasta el momento).

Otra particularidad es que la acción que tome un agente no solo se ve reflejada en ese instante, sino que en pasos siguientes, esta acción también tiene impacto.

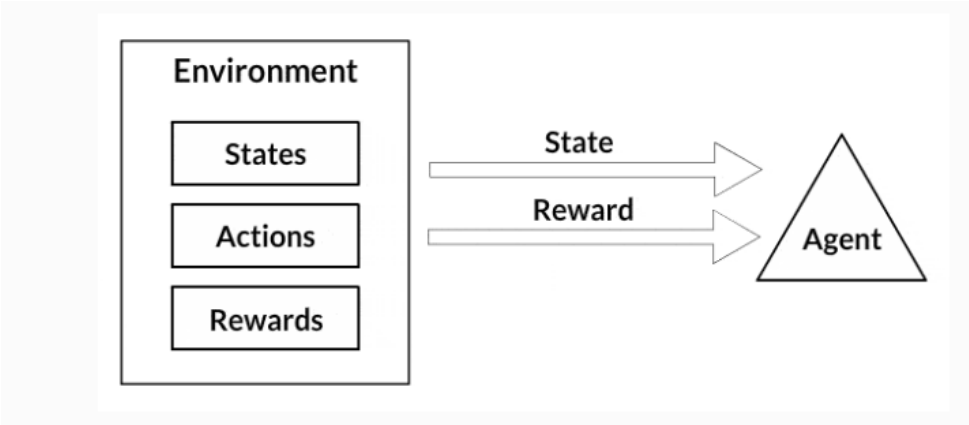


Figura 2: Proporciona información al ventilador del entorno.(Fuente: [8]).

En esta imagen **Figura** 2 observamos de lo que se nutre el agente. Este proceso permite que el agente aprenda qué acciones producen mejores resultados y, por tanto, cuáles debería repetir en situaciones similares. De esta manera, el ventilador aprende a actuar de forma óptima ajustando su velocidad en función del contexto, buscando siempre maximizar la eficiencia del flujo de aire en el túnel. El estado es la situación actual del sistema, las acciones serían las diferentes opciones para ajustar las velocidades y las recompensas nos indican el grado de acierto de cada acción.

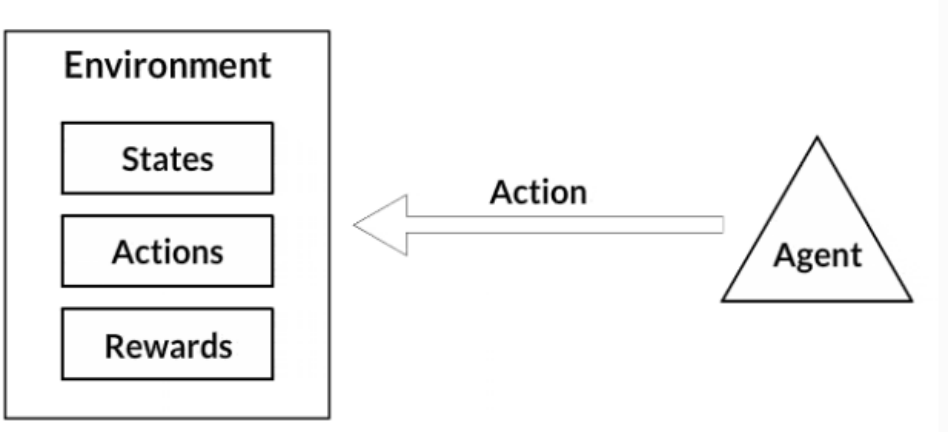


Figura 3: El ventilador ejecuta una acción (aumentar,mantener o disminuir la velocidad).(Fuente: [8]).

Por otro lado, en la imagen **Figura 3** el agente toma una decisión, es decir, ejecuta una acción, aumentar disminuir o mantener la velocidad de cada ventilador y actúa sobre el entorno. En este caso, el agente, que es el ventilador, toma una acción basándose en el estado actual del entorno.

La tabla Q es una estructura de datos que se utiliza para almacenar los datos que ha aprendido el agente durante el entrenamiento sobre lo buena que es una acción para un determinado estado. La tabla tiene un tamaño de todos los posibles estados que serían el número de filas por el número de acciones que serían las columnas.

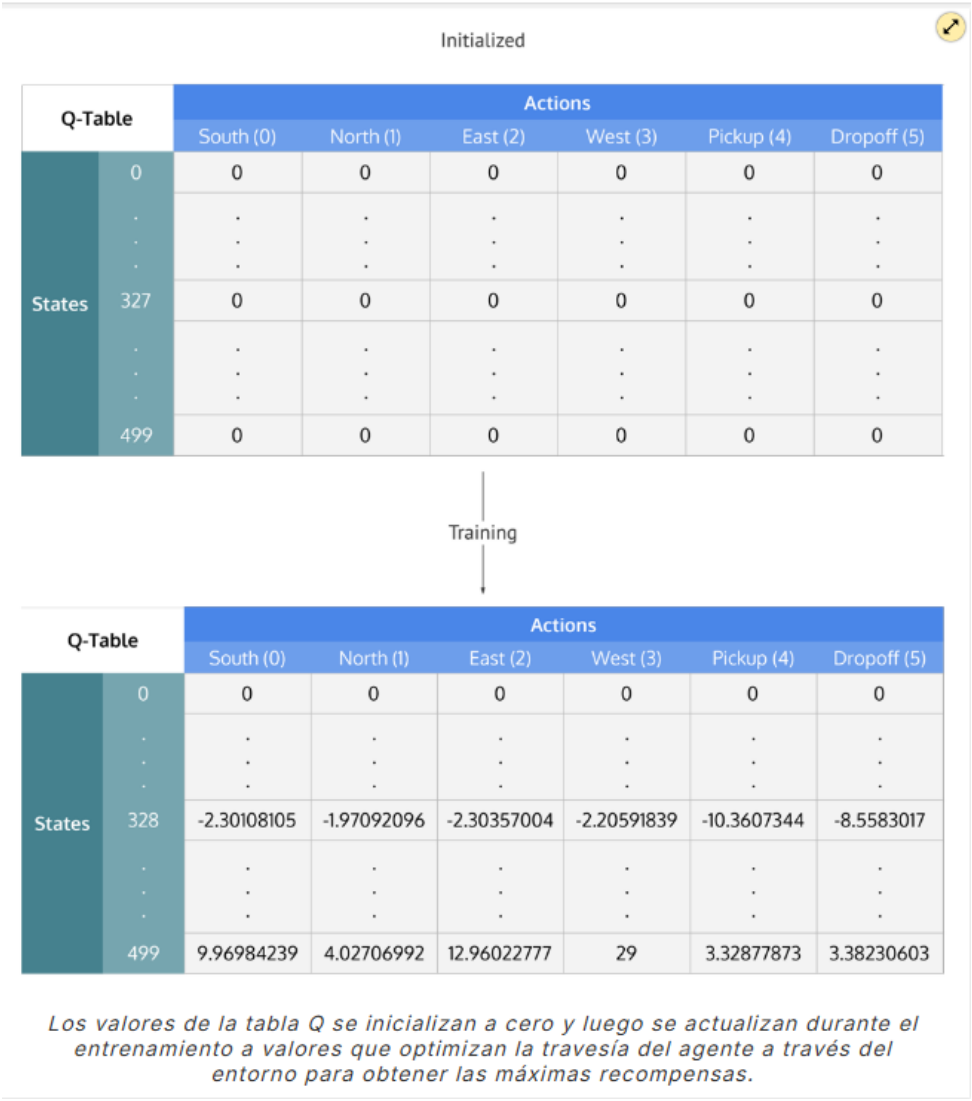


Figura 4: Ejemplo de una tabla Q.(Fuente: [8]).

Conforme se va iterando se actualiza la tabla Q que inicialmente se ha inicializado todo a ceros; después, con la experiencia que obtiene del entorno, recibe recompensas por las diferentes acciones y con esa información se incorpora a esta tabla para poder decidir

mejor en el futuro. El agente usa esta tabla para ver qué acción tomar.

Esta tabla se actualiza de la siguiente manera:

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q(s',a') \right) \tag{3}$$

Donde:

- α es la **tasa de aprendizaje** : Igual que en el aprendizaje supervisado, este parámetro determina el grado en que los valores Q se actualizan en cada iteración. Un valor alto de α implica que el agente aprende más rápido, pero puede generar oscilaciones(no converger), por otro lado un valor bajo hace que aprenda más lentamente pero de forma más estable(converge siempre).
- γ es el **factor de descuento** : Determina la importancia que se da a las recompensas futuras. Un valor alto incentiva al agente a buscar recompensas a largo plazo. Sin embargo, un valor bajo hace que el agente se centre únicamente en recompensas inmediatas, lo que puede hacerse encontrar subóptimos y quedarse atrapado.

Función de recompensa implementada

Para evaluar la calidad de las acciones tomadas por el agente, se ha implementado la siguiente función de recompensa: las velocidades objetivo deseadas menos las velocidades reales medidas. Esta diferencia se calcula utilizando el error cuadrático, penalizando así mayores desviaciones. La función de recompensa implementada es la siguiente:

$$r = -\frac{\sum \left(v_{\text{deseado}} - v_{\text{medido}} \right)^2}{24}$$

Donde:

- v_{deseado}: matriz de velocidades objetivo.
- v_{medido}: matriz de velocidades realmente alcanzadas por el ventilador.
- $\sum \left(v_{\text{deseado}} - v_{\text{medido}} \right)^2$: representa el error cuadrático total.
- El denominador 24: se utiliza para normalizar la recompensa dentro de un rango controlado, considerando un máximo de 24 celdas.

Esta función devuelve valores negativos, siendo 0 el valor máximo posible (cuando no hay error), y penaliza fuertemente aquellos estados en los que las velocidades medidas difieren considerablemente de las deseadas.

Resumen del proceso de aprendizaje Q-learning

A continuación se presenta un desglose paso a paso del funcionamiento del algoritmo Q-learning:

1. **Inicializar** la tabla Q con todos los valores en cero.
2. **Explorar acciones**: para cada episodio, seleccionar una acción entre todas las posibles para el estado actual S, aplicando una política de exploración (por ejemplo, ε-greedy).
3. **Ejecutar la acción** y transitar al nuevo estado S' como resultado de aplicar la acción a.
4. **Evaluar el nuevo estado**: para todas las acciones posibles desde el estado S', seleccionar la que tenga el valor Q más alto ($\max_{a'} Q(S', a')$).
5. **Actualizar** el valor de la tabla Q para el par (S, a) utilizando la fórmula de actualización:

$$Q(S,a) \leftarrow (1-\alpha) \cdot Q(S,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q(S', a') \right)$$

6. **Actualizar el estado actual**: asignar S' como el nuevo estado actual.
7. **Repetir**: si se alcanza el estado objetivo, finalizar el episodio y comenzar uno nuevo.

3.1.2. Limitaciones del algoritmo Q_learning en este problema

Al principio de esta sección se comentaba que la tabla $Q(s, a)$ se componía de dos dimensiones, donde cada fila corresponde a un estado del entorno y cada columna a una acción posible. Este enfoque resulta eficaz en entornos discretos y de baja complejidad, donde el número de estados y acciones es bajo.

Sin embargo, cuando se aplican a entornos con unos estados continuos o de alta dimensionalidad, deja de ser eficaz. En este caso, cada ventilador puede asumir infinitos valores de velocidad, lo que implica que el número de estados posibles es, en la práctica, infinito.

Como consecuencia, la tabla Q se vuelve inviable tanto desde el punto de vista del almacenamiento como del tiempo de entrenamiento. No es posible registrar o actualizar una entrada Q para cada posible combinación de velocidades, ya que esto requeriría manejar una cantidad astronómica de pares (s, a). Esta situación es conocida en la literatura como el *problema de la maldición de la dimensionalidad*.

Por tanto, estos problemas no pueden abordarse con Q_learning, y en su lugar, vamos a utilizar soluciones más escalables como las redes neuronales.

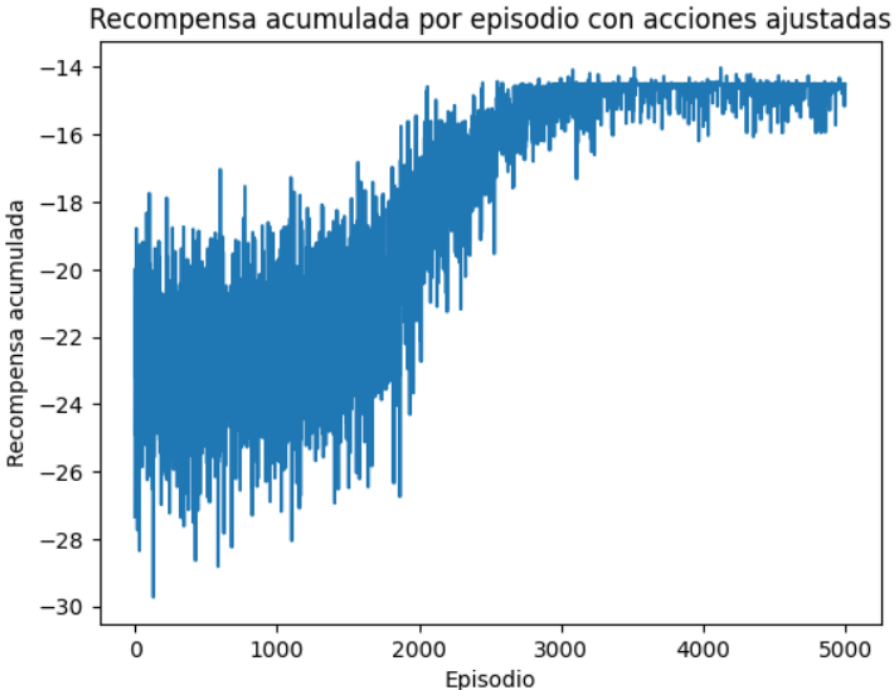


Figura 5: Resultados obtenidos con el método Q_learning

En la Figura 5 observamos cómo tiene una tendencia positiva al principio, pero tras muchos episodios se queda estancada en valores negativos.

3.2. DQN

3.2.1. Explicacion Algorimo

El algoritmo Deep Q-Network combina Q-learning con redes neuronales profundas. Este algoritmo utilizará redes neuronales para aproximar Q, es decir, no la predecirá. Esto permite poder manejar entornos con una gran dimensionalidad.

De hecho, contará con dos redes neuronales [9], la primera de ellas en la red neuronal principal(red Neural Network) aproxima la función $Q(s, a; \theta)$, donde θ representa los pará-

metros entrenables; esta red se encargará del entrenamiento con la experiencia del agente y la segunda de ellas es la red neuronal objetivo(target Neural Network) proporciona objetivos estables durante el entrenamiento mediante $Q(s', a'; \theta^-)$, donde θ^- son parámetros "congelados". Tiene la misma arquitectura que la red principal, solo que en este caso se actualizará cada cierto tiempo.

Además, también utiliza un buffer para almacenar experiencias vividas, entrena la red con estas muestras aleatorias, rompiendo la correlación y mejorando la estabilidad y fiabilidad.

El objetivo de este algoritmo, mediante el entrenamiento, es predecir el valor Q

$$Q^*(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a')\right]$$

donde:

- $Q^*(s, a)$ es el valor-Q óptimo
- $\gamma \in [0, 1)$ es el factor de descuento
- $\max_{a'} Q^*(s', a')$ representa el mejor valor futuro esperado

La función de Pérdida para las redes neuronales que queremos minimizar es la siguiente:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta^-)}_{\text{Target}} - \underbrace{Q(s, a; \theta)}_{\text{Predicción}}\right)^2\right]$$

Consiste en la minimización del error cuadrático entre las predicciones de la red principal y los objetivos generados por la red objetivo.

Donde:

- $Q(s, a; \theta)$: Valor-Q estimado por la **red neuronal principal** con parámetros θ.
- $Q(s', a'; \theta^-)$: Valor-Q estimado por la **red neuronal objetivo** (congelada), con parámetros θ^-.
- r: Recompensa inmediata recibida tras ejecutar la acción a en el estado s.
- γ: Factor de descuento, que pondera la importancia de las recompensas futuras.

3.2.2. Resumen algoritmo

1. **Inicialización:**

   a) Inicializamos tanto la red principal como la objetivo con la misma arquitectura:

   - Red principal $Q(s, a; \theta)$
   - Red objetivo $Q(s', a'; \theta^-)$ con pesos congelados

   b) Inicializar el buffer que guardará la experiencia $\mathcal{D}$ (*Replay Buffer*)

2. **Por cada episodio:**

   a) Inicializar el estado s

   b) Seleccionar una acción a usando ϵ-greedy.

   c) Ejecutar a, ver recompensa r y nuevo estado s'.

   d) Guardar (s, a, r, s') en el buffer $\mathcal{D}$.

   e) Coger una muestra aleatoria del buffer y calcular:

   $$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

   f) Calcular la función de pérdida:

   $$\mathcal{L}(\theta) = (y - Q(s, a; \theta))^2$$

   g) Actualizar los pesos θ mediante descenso por gradiente.

3. **Actualizar la red objetivo:**

   - Cada C pasos, copiar los pesos de la red principal:

   $$\theta^- \leftarrow \theta$$

4. **Reducir** ϵ: Disminuir de manera progresiva la tasa de exploración.

5. **Repetir**: Repetir el entrenamiento hasta encontrar convergencia o alcanzar el número de iteraciones máximo.

3.2.3. Limitaciones del algoritmo DQN en este problema

Aunque el algoritmo **Deep Q-Network (DQN)** ha demostrado ser exitoso en entornos complejos y discretos, en este caso, al problema de ajuste de velocidades de ventiladores no hemos obtenido buenos resultados.

Hemos encontrado varias limitaciones para nuestro problema:

1. **Los estados están mal definidos:** El estado de nuestro problema es una matriz continua de velocidades de 24 ventiladores. Sin embargo al aplanar la maztriz para introducirla en la tabla_Q se convierte en un vector unidimensional, por lo que perdemos la estructura espacial y la correlación entre los ventiladores vecinos, esto impide que la red neuronal aprenda patrones relevantes.

2. **Recompensas globales no específicas:** La función de recompensa utilizada penaliza el error cuadrático medio entre las velocidades deseadas y las medidas, es una medida global de todo el túnel de viento no proporciona información específica sobre qué ventilador debe ajustarse ni en dirección ni en magnitud.

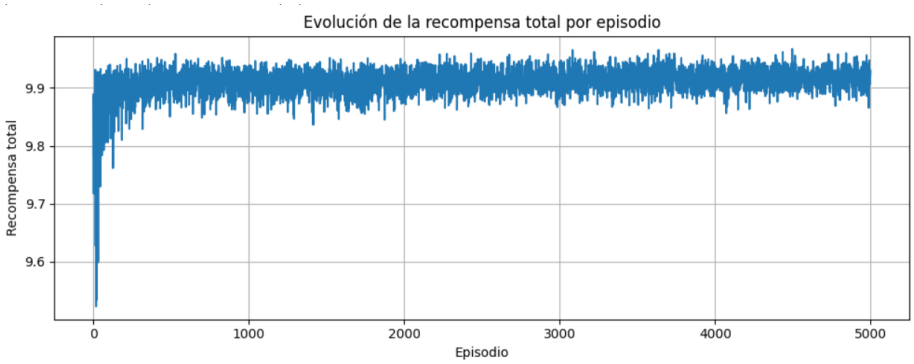


Figura 6: Resultados obtenidos con el método DQN.

En la **Figura 6** vemos cómo al principio consigue aprender rápidamente, pero se queda estancado en torno al 10 muy pronto, lo que nos puede hacer indicar que captura una solución trivial que maximiza la recompensa sin resolver el problema. Además, que se mantenga plana la recompensa nos hace pensar que no generalizará correctamente.

3.3. Metodología

3.3.1. Modelo Final

Introdución

El modelo que hemos implementado se basa en un problema estacionario simplificado que simula la propagación de la velocidad a lo largo de un túnel de viento. Este modelo se fundamenta en un proceso de difusión pura, descartando efectos temporales, cuyo

objetivo es predecir con el menor error posible las velocidades iniciales según un estado final deseado.

Para asegurar un proceso físico lógico, se impone el principio de la conservación de la energía cinética entre cada capa de difusión. Por lo tanto, aunque los valores de la velocidad se distribuyan espacialmente en cada una de las operaciones de convolución, la energía total del sistema se mantiene constante.

Este sistema nos permite generar datos sintéticos de alta calidad para entrenar nuestro modelo y es computacionalmente eficiente, estos datos son profundamente útiles para estudiar la inversa del problema, es decir, determinar los valores de velocidad en la entrada que producen un campo de velocidades deseado en la salida.

Formulación del modelo estacionario

La ecuación de difusión en tres dimensiones y en estado estacionario es la siguiente:

$$\nabla \cdot (D \nabla \vec{v}) = 0 \tag{4}$$

donde:

- $\vec{v} = v(i, j, k)$ representa la celda del campo escalar (i, j, k),
- D es el coeficiente de difusión,
- ∇ y $\nabla\cdot$ representan el gradiente y la divergencia, respectivamente.

Este modelo implica que el flujo de velocidad se difunde desde las regiones de mayor concentración hacia las de menor concentración hasta alcanzar un equilibrio, conservando la energía total del sistema.

Discretización del dominio

Consideramos un dominio discretizado con celdas indexadas por (i,j) en un plano paralelo a la pared del túnel de viento y capas indexadas por k. El campo de velocidad está dado por:

$$u_{i,j,k}, \tag{5}$$

Con la condición de frontera en la entrada $u_{i,j,0}$ = valor dado.

Convolución de propagación

La propagación del campo de velocidades se realiza de manera secuencial, aplicando una operación de difusión entre capas consecutivas en el eje z. Para cada paso $k \to k+1$, se emplea una operación de convolución 2D sobre la capa k, utilizando un *kernel* de difusión simétrico que representa la influencia de los vecinos:

$$u'_{i,j,k+1} = \sum_{m=-1}^{1} \sum_{n=-1}^{1} K_{m,n}\, u_{i+m,j+n,k}, \tag{6}$$

donde el *kernel* se define como:

$$K = \begin{pmatrix} K_{-1,-1} & K_{-1,0} & K_{-1,1} \\ K_{0,-1} & K_{0,0} & K_{0,1} \\ K_{1,-1} & K_{1,0} & K_{1,1} \end{pmatrix}. \tag{7}$$

El *kernel* está normalizado de tal forma que:

$$\sum_{m=-1}^{1} \sum_{n=-1}^{1} K_{m,n} = 1. \tag{8}$$

En este caso vamos a utilizar un kernel fijo de 3*3, que es el siguiente:

$$K = \begin{pmatrix} 0{,}05 & 0{,}1 & 0{,}05 \\ 0{,}1 & 0{,}45 & 0{,}1 \\ 0{,}05 & 0{,}1 & 0{,}05 \end{pmatrix}. \tag{9}$$

Este kernel respeta la simetría, además gran parte de la velocidad se mantiene en la celda central, mientras que el resto se difunde hacia las adyacentes.

Conservación de la energía

La energía cinética en una capa k se aproxima como:

$$E_k \approx \sum_{i,j} \left(u_{i,j,k}\right)^2. \tag{10}$$

Después del paso de convolución, la energía cinética provisional en la capa $k+1$ es:

$$E'_{k+1} \approx \sum_{i,j} \left(u'_{i,j,k+1}\right)^2. \tag{11}$$

Formulación del modelo estacionario

Tras cada convolución, la energía total puede variar. Para evitar esto, se aplica una normalización energética que ajusta la magnitud del campo de velocidades, preservando su distribución espacial, pero asegurando que la energía cinética total (suma de los cuadrados de las velocidades) se mantenga constante:

Normalización Post-procesamiento

Para conservar la energía cinética total al pasar de la capa k a la capa $k+1$, se calcula un factor de normalización N como:

$$N = \frac{E_k}{E'_{k+1}}. \tag{12}$$

El campo de velocidades final en la capa $k+1$ se obtiene mediante:

$$u_{i,j,k+1} = N\, u'_{i,j,k+1}. \tag{13}$$

Esto garantiza que:

$$\sum_{i,j} \left(u_{i,j,k+1}\right)^2 = E_k. \tag{14}$$

Esta normalización no garantiza que se mantenga la energía cinética total entre ventiladores contiguos, sino en el total del sistema, lo cual solo es una aproximación.

3.3.2. Conjunto de datos

Una vez definido el modelo físico de propagación mediante difusión, el siguiente paso será generar datos sintéticos que usaremos para entrenar nuestro modelo con redes neuronales. Esto implica crear pares de entrada-salida donde:

Entrada: Introduciremos una distribución de velocidades(simulando las velocidades de los ventiladores).

Salida: Obtenemos un campo de velocidades tras la última capa de propagación.

Para construir este conjunto de datos, necesitamos definir los siguientes parámetros que posteriormente optimizaremos:

Número de muestras: es el número de simulaciones independientes que se van a realizar.

Número de capas: Es la cantidad de capas de propagación K.

Tamaño de la malla: tamaño espacial de cada capa en el plano x*y.

Generamos entradas aleatorias, para ello, se genera una matriz del tamaño $v_0 \in \mathbb{R}^{\tilde{m}\times\tilde{n}}$, es decir, simularemos un túnel de viento de tamaño MxN con valores aleatorios (estos valores aleatorios simularán diferentes configuraciones de velocidades de ventiladores).

Aplicamos el modelo físico explicado anteriormente y obtenemos un vector con las velocidades en cada capa:

$$[v_0, v_1, \ldots, v_K]$$

donde v_K es la velocidad en la última capa.

Se almacena:

Entrada: v_0 (donde será un vector aplanado en una dimensión).

Salida: v_K (donde será un vector aplanado en una dimensión).

Esto forma un conjunto de datos de entrenamiento de la forma:

$$X = \{v_0^{(1)}, \ldots, v_0^{(n)}\} \in \mathbb{R}^{n\times(\tilde{n}^2)}$$
$$Y = \{v_K^{(1)}, \ldots, v_K^{(n)}\} \in \mathbb{R}^{n\times(\tilde{n}^2)}$$

Para representar el flujo de velocidades se discretiza como una malla bidimensional, es decir , una malla cuadrada de tamaño NxN.

Este conjunto de datos lo podemos usar para dos propósitos:

- Para entrenar un modelo directo que simule la propagación.
- Para entrenar un modelo inverso, que aprende a predecir la entrada v_0 que generaría una salida deseada v_K.

3.3.3. Modelo inverso

Una vez que se ha generado el dataset mediante el modelo físico, el siguiente paso es entrenar una red neuronal que actúe como modelo inverso. Esto significa:

> Queremos predecir la distribución inicial de velocidades v_0 que produce una salida deseada v_K tras la propagación por difusión.

En el modelo físico directo conocemos:

$$v_0 \xrightarrow{\text{modelo de difusión}} v_1, \ldots, v_K$$

Y podemos simular esta secuencia sin ambigüedad. Sin embargo, el proceso inverso es mucho más complejo, ya que:

- Puede haber múltiples configuraciones de v_0 que generen una misma v_K.
- El proceso de difusión pierde información (como ocurre en los sistemas físicos disipativos).
- El sistema no es analíticamente invertible, pero sí se puede aproximar mediante aprendizaje.

Objetivo del modelo inverso

Entrenar una red neuronal que, dada una velocidad deseada aguas abajo (en v_K), prediga qué configuración inicial de ventiladores v_0 podría haber generado ese estado.

$$\hat{v}_0 = f^{-1}(v_K) \tag{15}$$

donde f^{-1} representa la red neuronal entrenada para aproximar la inversión del modelo de difusión.

3.3.4. Resumen del modelo

El proceso de actualización del campo de velocidades desde la capa k hasta la capa $k+1$ se lleva a cabo en los siguientes pasos:

1. **Convolución provisional:**

$$u'_{i,j,k+1} = \sum_{m=-1}^{1} \sum_{n=-1}^{1} K_{m,n}\, u_{i+m,j+n,k} \tag{16}$$

2. **Cálculo de las energías cinéticas:**

$$E_k = \sum_{i,j} \left(u_{i,j,k}\right)^2, \quad E'_{k+1} = \sum_{i,j} \left(u'_{i,j,k+1}\right)^2 \tag{17}$$

3. **Factor de normalización:**

$$N = \frac{E_k}{E'_{k+1}} \tag{18}$$

4. **Normalización del las velocidades:**

$$u_{i,j,k+1} = N\, u'_{i,j,k+1} \tag{19}$$

3.3.5. Refinamiento de Parámetros

El rendimiento de un modelo depende en gran parte del ajuste de sus hiperparámetros. Un conjunto mal optimizado puede llevar a que no generalice correctamente, que presente problemas de sobreentrenamiento o infraajuste, y no sea capaz de capturar las relaciones relevantes en los datos.

Por ello, ahora vamos a encontrar la mejor combinación de parámetros, vamos a ver el efecto que produce de cada uno de los parámetros en nuestro modelo. Para conseguir esto, partimos de una arquitectura fija y vamos ajustando los parámetros uno a uno siguiendo una estrategia de optimización secuencial. Es decir, se mantienen constantes todos los parámetros salvo uno de ellos, de esta manera, veremos el efecto del parámetro en diferentes configuraciones del mismo.

Durante todo el proceso se han utilizado las siguientes métricas para evaluar el rendimiento de cada configuración y poder elegir la mejor:

$$\text{Error cuadrático medio (MSE)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\text{Raiz del Error cuadrático medio (RMSE)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\text{Error porcentual absoluto medio (MAPE)} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

$$\text{Coeficiente de determinación } (R^2) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}\left(y_i - \frac{1}{n}\sum_{j=1}^{n} y_j\right)^2}$$

$$\text{Similitud Coseno} = \frac{\vec{y}\cdot\hat{\vec{y}}}{\|\vec{y}\|\,\|\hat{\vec{y}}\|} = \frac{\sum_{i=1}^{n} y_i\hat{y}_i}{\sqrt{\sum_{i=1}^{n} y_i^2}\sqrt{\sum_{i=1}^{n}\hat{y}_i^2}}$$

Tipo de optimizador

Vamos a comenzar estudiando el tipo de optimizador, los optimizadores sirven para ajustar los pesos del modelo durante el entrenamiento. Cada optimizador utiliza una estrategia diferente para ajustar estos pesos. Para ello, vamos a evaluar los siguientes optimizadores:

Stochastic Gradient Descent (SGD)

Se diferencia del descenso por gradiente que en vez de utilizar todos los datos, solo coge una pequeña muestra para actualizar los pesos. De esta manera, es más eficiente computacionalmente. Fórmula:

$$\theta_{t+1} = \theta_t - \eta\nabla_\theta J(\theta)$$

donde:

- θ son los parámetros del modelo,
- η es la tasa de aprendizaje (*learning rate*),
- $J(\theta)$ es la función de pérdida.

Adagrad

Ajusta de manera dinámica el *learning rate* para cada parámetro durante el entrenamiento, si ha cambiado mucho últimamente le reduce la tasa de aprendizaje y si no se ha actualizado se lo incrementa.

Fórmula:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta J(\theta)$$

Donde:

$$G_t = \sum_{\tau=1}^{t} \nabla_\theta J(\theta_\tau)^2$$

RMSprop

Su funcionamiento es igual que en Adagrad, solo que en entrenamientos largos Adagrad disminuye demasiado, este optimizador solventa este problema.

Formula:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)\nabla_\theta J(\theta)^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla_\theta J(\theta)$$

Adaptive Moment Estimation (Adam)

Combina dos métodos momentum y RMSprop, quedándose con lo mejor de cada uno. Calcula un promedio de los gradientes y calcula un promedio de sus cuadrados de esta manera, adapta la tasa de aprendizaje.

Fórmula:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta))^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Nesterov-accelerated Adam (Nadam)

Combina Adam y Nesterov de manera que adapta la tasa de aprendizaje como Adam, pero además anticipa hacia dónde se dirige el gradiente, permitiendo realizar correcciones en cada paso.

Fórmula:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \left(\beta_1 \hat{m}_t + \frac{(1-\beta_1)}{1-\beta_1^t} \nabla_\theta J(\theta) \right)$$

Adamax

Es una variante de Adam que en vez de utilizar la segunda norma (cuadrados de los gradientes), utiliza la norma infinita (valor máximo absoluto)

Fórmula:

$$u_t = \text{máx}(\beta_2 u_{t-1}, |\nabla_\theta J(\theta)|)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} \hat{m}_t$$

A continuación se presenta la tabla donde se muestran los resultados obtenidos:

Optimizador	**MSE**	**RMSE**	**MAPE**	**R2 Score**	**Similitud Coseno**
Nadam	0.067336	0.259492	6.621194	0.191593	0.893281
Adam	0.069818	0.264231	6.829882	0.161781	0.889061
RMSprop	0.071737	0.267837	7.231384	0.138762	0.885866
Adamax	0.074485	0.272920	7.366648	0.105704	0.881152
Adagrad	0.097541	0.312316	7.495305	-0.171230	0.842668
SGD	0.181264	0.425751	4.265833	-1.175398	0.697433

Cuadro 1: Comparativa del desempeño del modelo con diferentes optimizadores.

Observamos en el **Cuadro 1** que en todas las medidas, excepto el MAPE, el mejor optimizador es Nadam, seguido muy de cerca por el optimizador Adam. Tanto para MSE como para RMSE son las que más cercanas a cero están, como para R^2 Score y similitud del coseno, cuyo objetivo es estar lo más próximo a 1.

learning rate

Este parámetro controla el tamaño de paso en que se actualizan los pesos durante el proceso de entrenamiento. Un valor bajo puede que avance demasiado despacio en el problema de optimización y se quede estancado en un óptimo local, mientras que si se elige un valor elevado de *learning rate* puede que el modelo no converja.

learning rate	MSE	RMSE	MAPE	R2 Score	Similitud Coseno
0.001	0.051831	0.227664	4.731510	0.378077	0.918866
0.0005	0.055689	0.235986	5.302935	0.331796	0.912522
0.005	0.060643	0.246257	5.120294	0.272277	0.904393
0.0001	0.062538	0.250077	5.759463	0.249626	0.901188
0.01	0.066533	0.257941	5.820329	0.201802	0.894533

Cuadro 2: Comparativa del desempeño del modelo con diferentes valores de *learning rate*.

Obtenemos que el mejor valor de *learning rate* es 0.001 donde se consigue el mejor resultado en las 5 medidas, seguido de 0.0005. En este caso, vemos que no es demasiado grande la diferencia entre los diferentes valores de *learning rate*.

Función de activación

Estas funciones de activación nos van a permitir aprender relaciones no lineales, sin ellas, solo tendríamos una relación lineal simple.

A continuación se presentan las definiciones matemáticas de las funciones de activación evaluadas en el modelo:

$$\text{ReLU}(x) = \text{máx}(0, x)$$

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\text{ELU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha(e^{x} - 1) & \text{si } x \leq 0 \end{cases}$$

Donde normalmente se usa $\alpha = 1$.

$$\text{SELU}(x) = \lambda \cdot \begin{cases} x & \text{si } x > 0 \\ \alpha(e^{x} - 1) & \text{si } x \leq 0 \end{cases}$$

Activación	MSE	RMSE	MAPE	R2 Score	Similitud Coseno
tanh	0.045757	0.213908	3.617030	0.450661	0.929028
elu	0.046198	0.214937	3.939938	0.445300	0.928139
selu	0.047000	0.216795	3.856820	0.435578	0.926791
relu	0.052166	0.228399	4.122425	0.373663	0.918387
sigmoid	0.058534	0.241939	4.589446	0.297209	0.907924

Cuadro 3: Comparativa del desempeño del modelo con diferentes funciones de activación

Obervamos en el **Cuadro 3** que la tangente hiperbólica obtiene los mejores resultados.

Número de muestras

Vamos a ver el número de muestras para el entrenamiento óptimo:

Nº Muestras	MSE	RMSE	MAPE	R2 Score	Similitud Coseno
5,000	0.053254	0.230768	15.446741	0.357799	0.916672
10,000	0.050526	0.224779	3.654597	0.394758	0.921042
20,000	0.045826	0.214069	3.997266	0.450823	0.928865
50,000	0.041056	0.202624	3.049625	0.507001	0.936316
100,000	0.041268	0.203146	3.567800	0.504221	0.936095
200,000	0.036091	0.189977	3.546603	0.566995	0.944330
300,000	0.035219	0.187667	3.087349	0.577284	0.945720
400,000	0.033349	0.182618	3.034292	0.599957	0.948685
800,000	0.031166	0.176538	5.346358	0.625961	0.952181
1,000,000	0.024707	0.157184	3.026556	0.703443	0.962260

Cuadro 4: Comparativa del desempeño del modelo con diferentes número de muestras

Como observamos en el **Cuadro 4**, cuanto más muestras tiene nuestro modelo mejores resultados obtenemos en todas las medidas. Sin embargo, hay que tener cuidado con el sobreentrenamiento, que nos puede producir excelentes resultados en el entrenamiento pero malos datos en test, es decir, que no generaliza bien. Además, el coste computacional aumenta de manera considerable, por lo que cogeremos un valor arbitrario de 200.000 muestras.

A continuación tenemos los resultados obtenidos:

Número de capas ocultas y neuronas

Ahora vamos a considerar el número de capas ocultas y el número de neuronas por pares, es decir, vamos a evaluar combinaciones de ambas variables, debido a que estos dos parámetros están muy relacionados y en conjunto, nos pueden ofrecer mayor información.

Capas × Neuronas	MSE	RMSE	MAPE	R2 Score	Similitud Coseno
2 × 256	0.039761	0.199402	14.016322	0.522363	0.938411
1 × 256	0.039894	0.199734	13.055314	0.520745	0.938182
1 × 128	0.041124	0.202790	12.317664	0.505967	0.936207
3 × 128	0.042021	0.204990	12.990236	0.495215	0.934768
4 × 256	0.043000	0.207365	13.660144	0.483437	0.933219
2 × 64	0.043140	0.207701	12.453571	0.481778	0.933024
5 × 128	0.043343	0.208189	15.839331	0.479327	0.932644
1 × 64	0.043516	0.208605	15.490810	0.477243	0.932368
3 × 256	0.043752	0.209171	12.473093	0.474421	0.931998
5 × 256	0.044034	0.209843	12.496070	0.471039	0.931542
3 × 64	0.044397	0.210705	14.741740	0.466685	0.930952
2 × 128	0.044834	0.211741	15.473109	0.461375	0.930239
4 × 128	0.045296	0.212830	14.103507	0.455841	0.929492
5 × 64	0.045885	0.214209	13.594835	0.448771	0.928552
4 × 64	0.046493	0.215622	15.397469	0.441475	0.927555
1 × 32	0.056856	0.238444	17.206343	0.316933	0.910674
2 × 32	0.056911	0.238560	17.152849	0.316287	0.910589
4 × 32	0.057646	0.240096	16.429696	0.307447	0.909318
3 × 32	0.057744	0.240299	15.846581	0.306298	0.909168
5 × 32	0.060284	0.245527	13.219647	0.275766	0.904971

Cuadro 5: Comparativa del desempeño del modelo según el número de capas y neuronas por capa.

Observamos en el **Cuadro 5** que la mejor opción son dos capas y 256 neuronas, entre los mejores modelos tambien vemos que hay muchos de ellos que son sencillos, ya que los mejores resultados se obtienen combinaciones con pocas capas ocultas, por otro lado el número de neuronas sí es óptimo un gran número de ellas.

Batch

El batch son el conjunto de muestras que se procesan en una sola iteración, es decir, que no se coge una muestra en cada iteración, sino que se coge un lote de ellas.

A continuación vamos a estudiar cuáles son las que nos dan mejores resultados:

Batch Size	MSE	RMSE	MAPE	R2 Score	Similitud Coseno
16	0.034411	0.185501	9.544038	0.587127	0.947027
32	0.037700	0.194164	10.514024	0.547648	0.941813
64	0.039734	0.199333	7.538444	0.523278	0.938573
128	0.041098	0.202728	4.688606	0.506895	0.936543
256	0.043443	0.208431	4.164542	0.478817	0.932621

Cuadro 6: Comparativa del desempeño del modelo según el tamaño del *batch*.

Observamos en el **Cuadro 6** que cuanto más pequeñas son las muestras de los lotes, obtenemos mejores resultados por tanto, la mejor solución es 16.

Número épocas

El número de épocas son las veces que se recorre todo el conjunto de entrenamiento durante el aprendizaje.

Épocas	MSE	RMSE	MAPE	R2 Score	Similitud Coseno
50	0.037772	0.194350	3.358280	0.546450	0.941642
100	0.032366	0.179906	3.111474	0.611445	0.950226
200	0.027521	0.165893	2.754123	0.669564	0.957828
500	0.020453	0.143015	2.424844	0.754438	0.968833

Cuadro 7: Desempeño del modelo según el número de épocas de entrenamiento.

Se muestra en el **Cuadro 7** una mejora consistente en el rendimiento del modelo conforme se aumentan las épocas. Para esto se ha realizado una técnica de *Stop early* que consiste en parar el aprendizaje si no se mejora el valor de "val_loss" que es el criterio que mide el error en el conjunto de validación. Por tanto, se parará si durante 20 iteraciones sucesivas no mejora este criterio. Una vez se detiene la ejecución, el modelo vuelve a los pesos que obtuvieron la mejor pérdida de validación.

3.3.6. Resultados

Una vez optimizado el modelo y entrenado, vamos a evaluar la capacidad del modelo de predecir las velocidades iniciales correctamente según las velocidades deseadas al final del túnel de viento. Para evaluar la capacidad del modelo, vamos a ver su comportamiento tanto en un conjunto de validación como en un conjunto test.

Para ello se seleccionan unas velocidades finales deseadas que anteriormente se han

generado en el dataset de entrenamiento. Este modelo generará una predicción de las velocidades iniciales según el aprendizaje realizado.

Una vez se obtienen las velocidades iniciales predichas por el modelo, se aplica el modelo de difusión a estas velocidades, propagándose las velocidades por las capas. De esta manera, obtenemos la velocidad final generada por nuestras velocidades iniciales predichas.

En cada ejemplo, vamos a tener 3 gráficos el primero de ellos representará la velocidad deseada, el segundo de ellos representará la velocidad final conseguida por las velocidades iniciales predichas por nuestro modelo y por último, tendremos un gráfico con la diferencia de velocidades entre las deseadas y las obtenidas, es decir, la diferencia entre el primer gráfico y el segundo.

Figura 7: Escala vridis.

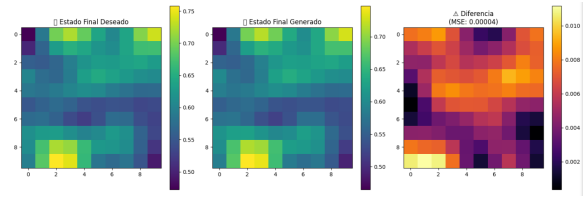
Figura 8: Escala inferno.

Por otro lado, se está utilizando la paleta viridis, como vemos en la **Figura 7** por lo que el color azul oscuro corresponde a velocidades más bajas, valores moderados tendrán un color verde y los valores más altos tienen un color amarillento. Respecto al gráfico de las diferencias entre las velocidades obtenidas y deseadas, se utiliza una paleta llamada inferno. Observando la **Figura 8**, si el color es negro, la diferencia es prácticamente nula. Si es rojo, una diferencia moderada y amarilla una diferencia significativa dentro de la escala. A la derecha de cada gráfico tienen respectivamente su escala correspondiente.

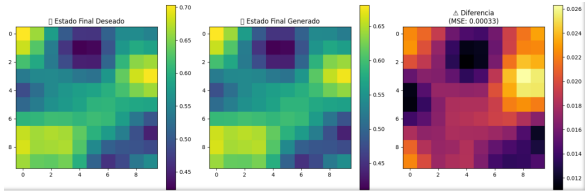
Vamos a comenzar viendo ejemplos en el conjunto de validación.

Resultados obtenidos en el conjunto de validación

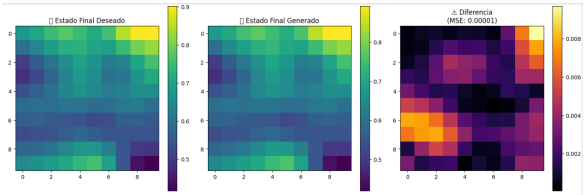
A continuación vamos a ver 6 ejemplos:



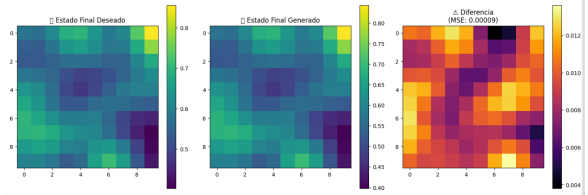
(a) Ejemplo 1



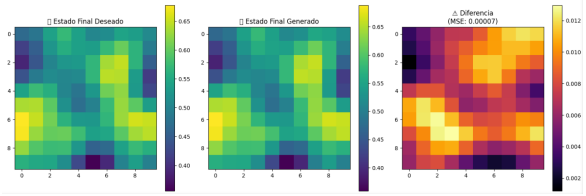
(b) Ejemplo 2



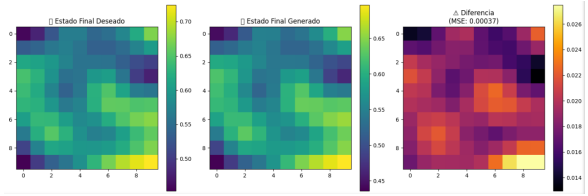
(c) Ejemplo 3



(d) Ejemplo 4



(e) Ejemplo 5



(f) Ejemplo 6

Figura 9: Comparación visual de diferentes ejemplos generados por el modelo. Cada ejemplo muestra el estado deseado, el generado y la diferencia entre ambos.

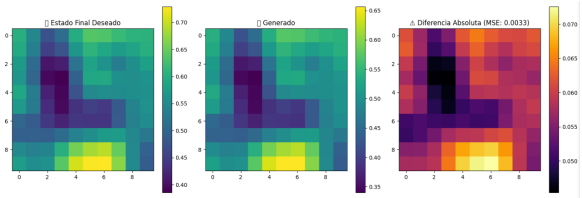
Ejemplo	**MSE**
Ejemplo 1	0.00004
Ejemplo 2	0.00033
Ejemplo 3	0.00001
Ejemplo 4	0.00009
Ejemplo 5	0.00007
Ejemplo 6	0.00037

Cuadro 8: Error cuadrático medio (MSE) de cada uno de los ejemplos

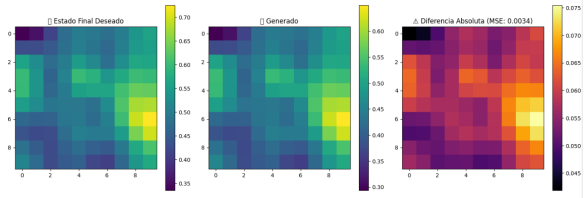
Vemos que obtenemos un MSE medio de los cinco ejemplos alrededor de 0.0005, por lo que obtenemos una muy buena aproximación de las velocidades iniciales. Observamos gráficamente que se parecen las dos primeras gráficas y logra capturar las velocidades más extremas, es decir, las que están cerca de 0 y de 1. En cuanto al tercer gráfico, observamos cómo hay algunas partes que tienen color amarillo, pero no es predominante y además se manejan escalas muy pequeñas.

Resultados obtenidos en el conjunto de Test

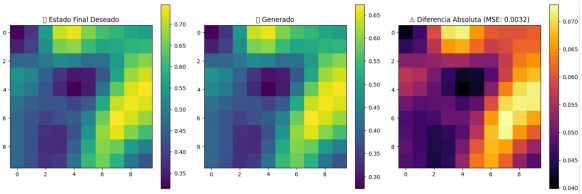
Vamos a continuar viendo la precisión que tiene nuestro modelo en test, como ya sabemos lo más importante de un modelo es su poder de generalización. Para ello necesitamos ejemplos con los que no haya entrenado el modelo. A continuación vamos a ver 6 ejemplos.



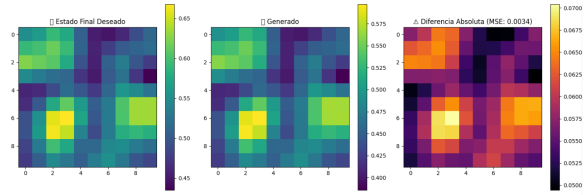
(a) Ejemplo 1 - Test



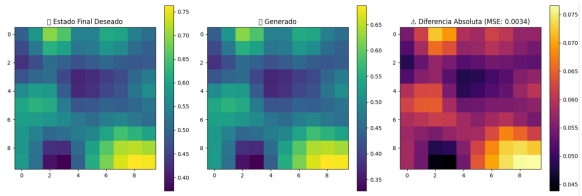
(b) Ejemplo 2 - Test



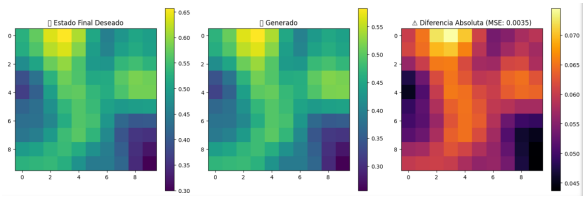
(c) Ejemplo 3 - Test



(d) Ejemplo 4 - Test



(e) Ejemplo 5 - Test



(f) Ejemplo 6 - Test

Figura 10: Visualización de los resultados del modelo inverso sobre ejemplos del conjunto de prueba.

Ejemplo de Test	**MSE**
Ejemplo 1	0.0033
Ejemplo 2	0.0034
Ejemplo 3	0.0032
Ejemplo 4	0.0034
Ejemplo 5	0.0034
Ejemplo 6	0.0035

Cuadro 9: Error cuadrático medio (MSE) entre el estado final deseado y el generado en cada ejemplo del conjunto de test.

Como observamos en la **Tabla 9**, se reduce el MSE respecto a los ejemplos de validación considerablemente, sin embargo, se siguen obteniendo excelentes resultados. En cuanto a las gráficas, observamos cómo “a simple vista” no se observan grandes diferen-

cias, y en cuanto a la última gráfica, obtenemos resultados similares a los de validación, aunque, teniendo en cuenta la escala, son algo mayores.

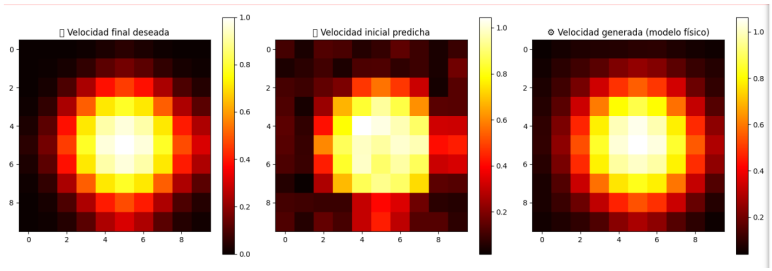
Resultados obtenidos en formas especiales

Por último, vamos a medir el rendimiento de nuestro modelo con diferentes distribuciones espaciales, vamos a ver los siguientes patrones: círculo, triángulo, cruz y ondas. Esto nos va a permitir ver, en diferentes escenarios, el comportamiento del modelo de forma visual.

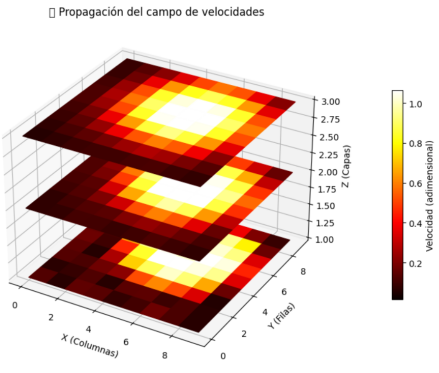
Vamos a tener diferentes gráficos. El primero de ellos se divide en tres subgráficos. El primero de ellos será la velocidad deseada, el segundo, las velocidades iniciales predichas y el último subgráfico nos mostrará las velocidades generadas. El segundo gráfico nos va a permitir ver la propagación de las velocidades por las diferentes capas ocultas del modelo y en el tercer gráfico veremos las diferencias entre las velocidades deseadas y predichas en cada capa oculta. En el tercer gráfico se utiliza la paleta inferno explicada anteriormente, en los resultados obtenidos del conjunto de validación y del conjunto test. En cuanto a los 2 primeros gráficos, se utiliza la escala hot que observamos en la **Figura 11**.

Figura 11: Escala inferno.

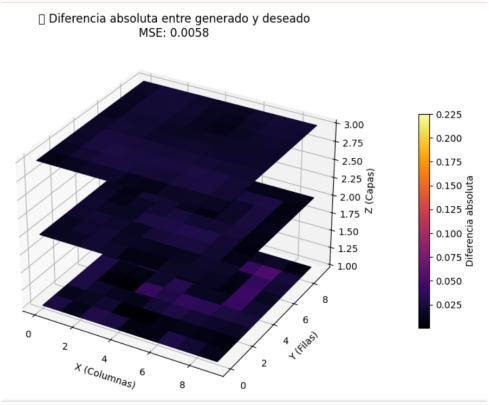
Círculo



(a) Estado de las diferentes velocidades



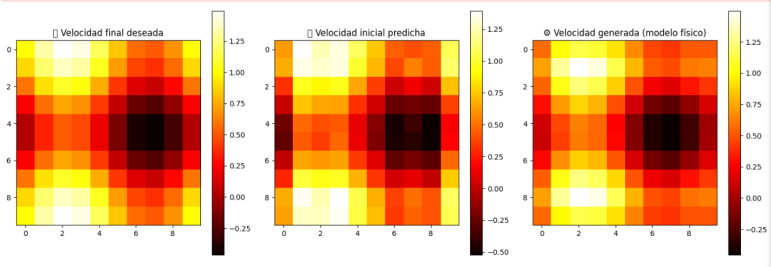
(b) Propagación en las capas ocultas



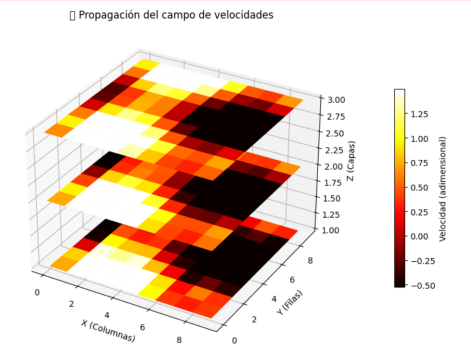
(c) Diferencia respecto a la deseada

Figura 12: Visualización del experimento con forma de Círculo: estado deseado, propagación y diferencia.

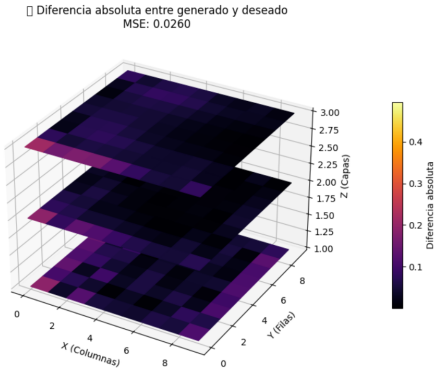
Ondas



(a) Estado de las diferentes velocidades



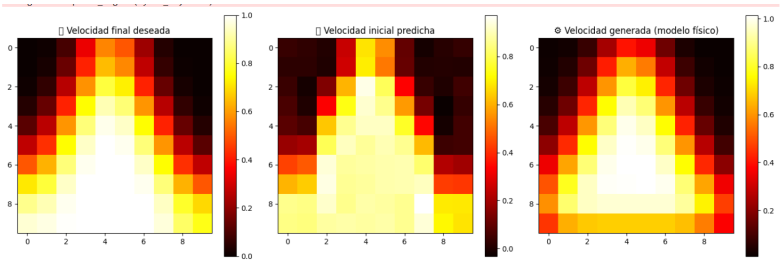
(b) Propagación en las capas ocultas



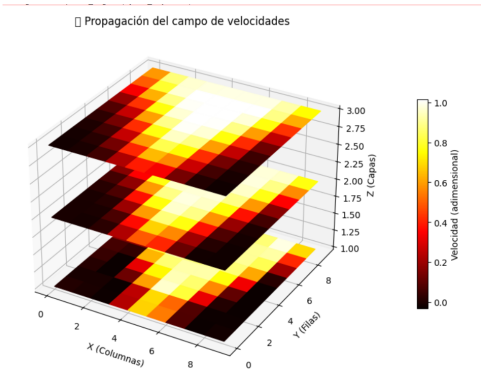
(c) Diferencia respecto a la deseada

Figura 13: Visualización del experimento con forma de Ondas: estado deseado, propagación y diferencia.

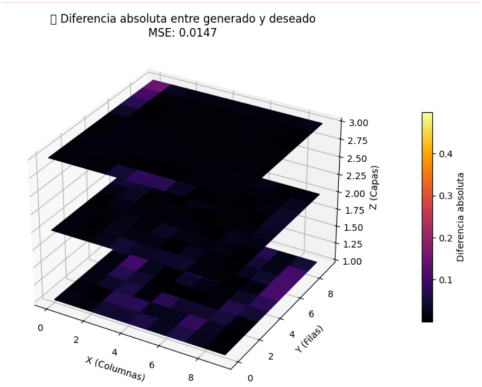
Triangulo



(a) Estado de las diferentes velocidades



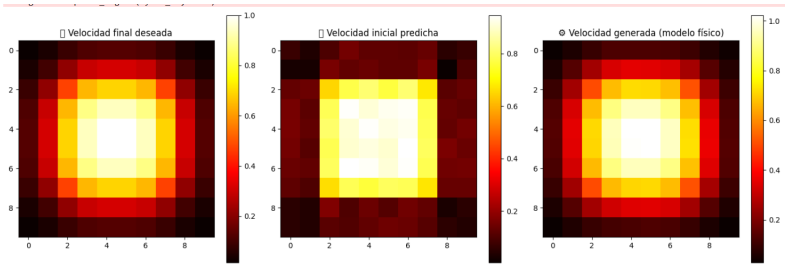
(b) Propagación en las capas ocultas



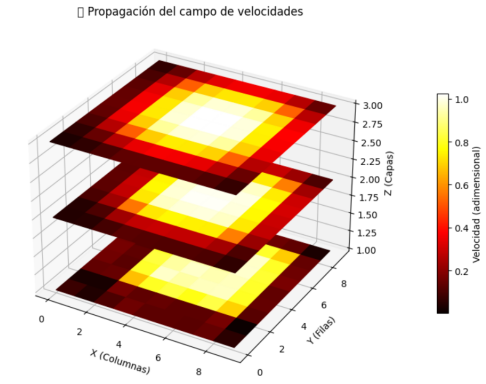
(c) Diferencia de las velocidades deseadas a las predichas

Figura 14: Visualización del experimento con forma de Triangulo: estado objetivo, propagación y diferencia.

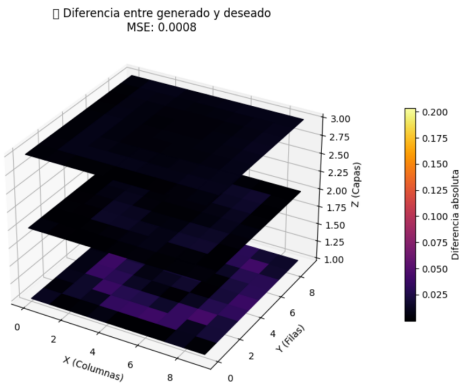
Cuadrado



(a) Estado de las diferentes velocidades



(b) Propagación en las capas ocultas



(c) Diferencia respecto a la deseada

Figura 15: Visualización del experimento con forma de cuadrado: estado objetivo, propagación y diferencia.

Diferentes formas	**MSE**
Circulo	0.0058
Triangulo	0.0147
Ondas	0.0260
Cuadrado	0.0008

Cuadro 10: Error cuadrático medio entre el estado final deseado y el generado en las ditintas formas.

En todas ellas, como vemos en la **Tabla** 10 obtenemos un MSE bajo, por lo que el modelo logra aprender estructuras regulares. Sin embargo, se aprecia una reducción del MSE en aquellos que tienen unas transiciones más bruscas, como por ejemplo el triángulo y obtiene los mejores resultados en aquellos que tienen transiciones menos bruscas como es el caso del círculo con un 0.0058 de MSE. También observamos de manera visual en el gráfico de las diferencias correspondiente al apartado c) de las **Figuras** 12, 13, 14 y 15 cómo predomina el color azul oscuro, por lo que obtenemos diferencias pequeñas en la mayoría de los puntos de las velocidades.

4. Conclusiones y lineas futuras

En este trabajo se han desarrollado diferentes tipos de modelos. Hemos comenzado implantando modelos basados en aprendizaje por refuerzo, primero mediante el Q_learning y posteriormente el DQN. Tras no conseguir los resultados esperados con estos modelos, se ha desarrollado un modelo basado en redes neuronales.

Para ello, se ha realizado un modelo físico estacionario para simular el comportamiento del túnel de viento, este modelo consiste en un modelo de difusión, en el que cada ventilador se ve influenciado por las velocidades vecinas. Estas velocidades se ajustan mediante el kernel, además, en cada paso de propagación se debe normalizar para que la energía total del sistema se conserve a lo largo del túnel de viento.

Una vez desarrollado este modelo, se ha desarrollado el enfoque opuesto, ya que el objetivo del trabajo no es predecir las velocidades al final del túnel de viento, sino que, con unas velocidades finales deseadas, predecir qué velocidades iniciales son las idóneas para conseguir esas velocidades deseadas.

Una vez construido el modelo, se ha realizado una optimización de los parámetros para conseguir el menor error posible. Además, se ha llevado a cabo una técnica de early stopping y checkpoint para evitar el sobreaprendizaje y no aumentar de manera excesiva el coste computacional.

Tras estos ajustes, se ha llevado a cabo la evaluación del modelo mediante un conjunto de validación y un conjunto test, en ambos se ha conseguido obtener buenos resultados, con una media del error de 0.0005 y 0.00295, respectivamente.

Además se ha experimentado también con diferentes formas regulares, donde se ha demostrado que el modelo generaliza correctamente obteniendo valores de MSE bajos y a través de una visualización de imágenes hemos visto que las diferencias que hay entre las velocidades deseadas y las generadas tienen un error mínimo. Sin embargo, se ha apreciado que en aquellas figuras que tienen formas abruptas, como son el triángulo o las ondas, se obtienen peores resultados que en aquellas que son cambios suaves como el cuadrado o el círculo.

Por tanto, se ha demostrado la utilidad de las redes neuronales y el aprendizaje automático para este tipo de problemas. Este trabajo sienta las bases sobre otro tipo de mejoras que se van a comentar a continuación.

Se puede realizar un modelo no estacionario que tenga en cuenta la evolución temporal del flujo dentro del túnel de viento. También se pueden tomar datos reales experimentales, es decir, incorporar al túnel de viento puntos de medición. Otro factor que se puede implementar es añadir otros fenómenos físicos como la presión.

Puedes consultar el código fuente en el repositorio Tunel-de-Viento.

Referencias

[1] Bruno Chanetz. A century of wind tunnels since Eiffel. *Comptes Rendus Mécanique*, 345(8):581–594, 2017. https://doi.org/10.1016/j.crme.2017.05.012.

[2] J. M. M. Barata y F. M. S. P. Neves. *The history of aviation education and training.* Open Journal of Applied Sciences, 7(4):196–205, 2017. https://doi.org/10.4236/ojapps.2017.74017.

[3] P. V. S. Souza y P. M. C. de Oliveira. "Numerical wind tunnels". Preprint en arXiv:1509.06794, 2015. Disponible en: https://arxiv.org/abs/1509.06794.

[4] V. A. Vlasov. *Simplified diffusion model of gas hydrate formation from ice. International Journal of Heat and Mass Transfer*, 165 Part B:120701, February 2021. https://doi.org/10.1016/j.ijheatmasstransfer.2020.120701.

[5] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis y R. Willett. *Deep Learning Techniques for Inverse Problems in Imaging.* Preprint arXiv:2005.06001, 2020. Disponible en: https://arxiv.org/abs/2005.06001.

[6] C. Cheng y G.-T. Zhang. *Deep Learning Method Based on Physics Informed Neural Network with Resnet Block for Solving Fluid Flow Problems. Water*, 13(4):423, 2021. https://doi.org/10.3390/w13040423.

[7] Jesse Clifton y Eric Laber. Q-Learning: Theory and Applications. *Annual Review of Statistics and Its Application*, 7:279–301, 2020. https://doi.org/10.1146/annurev-statistics-031219-041220.

[8] LearnDataSci.com. Reinforcement Q-Learning from scratch in Python: OpenAI Gym. Consultado el 27 de mayo de 2025. Disponible en: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/.

[9] H. van Hasselt, A. Guez y D. Silver. *Deep Reinforcement Learning with Double Q-Learning.* Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 2094–2100, 2016. https://doi.org/10.1609/aaai.v30i1.10295.