Redes neuronales y algoritmos evolutivos para jugar Chrome Dino

Alejandro Calgaro, Agustín Chen
Estudiante de Ingeniería en Informática, alejandrocalgaro@gmail.com

Resumen*—En este proyecto para la asignatura Inteligencia Computacional se desarrolló una inteligencia artificial capaz de aprender a jugar al videojuego Chrome Dino o T-rex runner. La importancia de este trabajo se encuentra en la aplicación de técnicas de inteligencia computacional en el contexto de un juego dinámico, en el cual la complejidad surge de la naturaleza no determinística del juego, donde los cambios de velocidad y la variabilidad de los obstáculos requieren respuestas rápidas y adaptativas, poniendo a prueba la capacidad de aprendizaje de los modelos desarrollados. Se propuso implementar dos soluciones al problema, una utilizando redes neuronales entrenando un perceptrón multicapa (MLP) con gradiente y otra empleando algoritmos evolutivos. Sobre estas soluciones se realizaron diversas pruebas para determinar la arquitectura y parámetros óptimos a utilizar. Ambas soluciones implementadas lograron una efectividad considerablemente mayor a la obtenida por un ser humano durante el juego. Finalmente se realizó la comparación entre ambas soluciones y se determinó que con el mejor algoritmo evolutivo se obtiene una mejor solución que con el mejor MLP.

***Palabras clave*—redes neuronales, aprendizaje supervisado, gradiente, algoritmos evolutivos.**

I. INTRODUCCIÓN

En el ámbito de la inteligencia computacional en videojuegos, uno de los desafíos interesantes que se pueden analizar es otorgar a una inteligencia artificial la capacidad de dominar el juego, que en el caso de este trabajo será el videojuego Chrome Dino, también conocido como T-Rex runner. En éste juego, el personaje controlado por el usuario es un dinosaurio y existen obstáculos en forma de cactus o pájaros, y el objetivo es que el personaje logre evitar la mayor cantidad de obstáculos posibles para avanzar en el camino y aumentar su puntuación, la cual aumenta por cada paso dado por el personaje. En caso de chocar contra uno de los obstáculos, se pierde el juego. Al iniciar una partida, el personaje comienza corriendo y las únicas acciones posibles son saltar, agacharse o no hacer nada, es decir, seguir corriendo.

Si bien es un juego simple, al ser un juego dinámico donde la velocidad del juego aumenta con el tiempo y los obstáculos aparecen de forma aleatoria, se vuelve interesante la aplicación de técnicas de inteligencia computacional ya que se requieren respuestas rápidas y adaptativas, lo cual significa poner a prueba la capacidad de aprendizaje de la inteligencia artificial en tiempo real.

Analizando trabajos similares nos encontramos por un lado la utilización de redes neuronales artificiales mediante un aprendizaje supervisado [1], realizando una comparación directa entre el resultado obtenido por la red y el resultado esperado, utilizando un algoritmo de optimización basado en el descenso del gradiente como el algoritmo de retropropagación, para ir ajustando iterativamente los pesos entre las conexiones de las neuronas y minimizar el error.

Otro enfoque posible [2] es utilizar el aprendizaje por refuerzo, cuyo objetivo es que la red neuronal aprenda a través de la interacción con el entorno (el juego) y se adapte a situaciones cambiantes, aprendiendo qué hacer para maximizar la puntuación, que en este caso sería logrando que el modelo aprenda a descubrir una manera de evitar los obstáculos.

Por otra parte, en [1] y [3] se habla de optimizar el diseño y los parámetros de las redes neuronales artificiales mediante el uso de algoritmos evolutivos, creando una población de individuos donde cada individuo representa una solución al problema, y se busca evolucionar esta población para ir obteniendo cada vez individuos más aptos, sin depender del conocimiento de un experto que le indique las acciones que debe realizar.

En el contexto de la inteligencia artificial en videojuegos, como se menciona en [4], las entradas a la red neuronal toman datos directamente del entorno del juego, en forma de datos numéricos, mientras que la salida, que podría ser más de una, se utiliza para determinar la acción que se debería llevar a cabo.

Finalmente, en [5] se puede observar como se ha realizado un análisis y resolución de otros tipos de juegos o problemas, utilizando las mismas técnicas antes mencionadas y realizando diversos experimentos para obtener una comparación con distintos algoritmos o arquitecturas de red neuronal, buscando encontrar la mejor solución posible para cada problema.

Respecto a las soluciones propuestas en este trabajo, se utilizó por un lado el entrenamiento supervisado de un perceptrón multicapa (MLP) mediante gradiente, y por otra parte una solución empleando algoritmos evolutivos. En ambas soluciones se propuso tomar como entradas a una red neuronal datos del juego como la distancia del dinosaurio al próximo obstáculo, velocidad del juego, coordenada Y del dinosaurio, altura, ancho y coordenada Y del obstáculo.

Una vez implementadas las propuestas de resolución del problema, se realizaron diversos experimentos para obtener los valores óptimos para la arquitectura de red neuronal y parámetros de la red y del algoritmo evolutivo, además de realizar una comparativa entre las soluciones implementadas y entre la efectividad de las soluciones obtenidas frente a la que podría lograr un ser humano jugando.

II. RESOLUCIÓN

La resolución del problema se dividió en dos partes, por un lado, implementar una red neuronal que aprenda a jugar utilizando un algoritmo de propagación hacia atrás junto con aprendizaje supervisado y, por otro lado, implementar un algoritmo evolutivo en el cual los individuos a través de las generaciones puedan ir adaptando los pesos a ser utilizados en la red neuronal para aprender a evitar los obstáculos del juego.

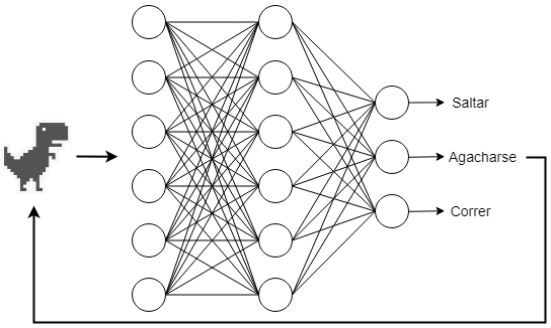

Fig. 1: Diagrama de solución mediante red neuronal

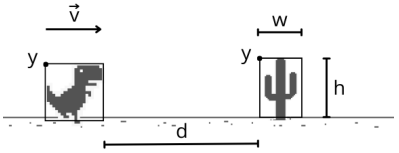

Fig. 2: Entradas a la red neuronal

En la Fig.1 se muestra un diagrama general que resume la propuesta de red neuronal que se utilizó para resolver el problema. Se utilizó una red neuronal con una capa de entrada que cuenta con 6 neuronas y que recibe como entradas la distancia del dinosaurio al próximo obstáculo, velocidad del juego, coordenada Y del dinosaurio, altura, ancho y coordenada Y del obstáculo, como se puede observar en la Fig. 2. Luego, una capa oculta en la cual se ha probado distinta cantidad de neuronas y se llegó a la conclusión de que la mejor alternativa es utilizar 6 neuronas, y una capa de salida con 3 neuronas, para que el dinosaurio pueda tomar tres posibles decisiones, saltar, agacharse o seguir corriendo.

A. Redes neuronales

Centrándonos en los detalles de la resolución utilizando solo redes neuronales, se realizó un entrenamiento supervisado de un perceptrón multicapa, comenzando con una inicialización al azar de los pesos de la red, aplicando luego la propagación hacia adelante de cada patrón de entrenamiento, utilizando en este caso como función de activación no lineal la función sigmoidea, hasta obtener la salida final de la red, la cual se compara con la salida deseada para calcular el error cometido y aplicar luego la propagación hacia atrás, donde gracias a la retropropagación del gradiente de error se puede finalmente realizar el ajuste de los pesos para que la red pueda aprender de los datos de entrenamiento.

Para comprobar el funcionamiento durante su entrenamiento se realiza una etapa de validación o monitoreo en la cual, utilizando un dataset de validación, se comprueba la cantidad de errores de la red neuronal al clasificar estos patrones según las tres salidas posibles. De esta manera se continúa entrenando hasta lograr una tasa de error aceptable o un número máximo de iteraciones.

B. Algoritmos evolutivos

Como segunda parte del trabajo, se implementó el uso de algoritmos evolutivos para la resolución del problema, donde cada individuo de la población será un dinosaurio del juego que tendrá asociado los pesos de una red neuronal que le permitirá tomar decisiones dentro del juego, teniendo en cuenta además la puntuación que ese individuo alcanzó en el juego para ser utilizada como valor de fitness que nos

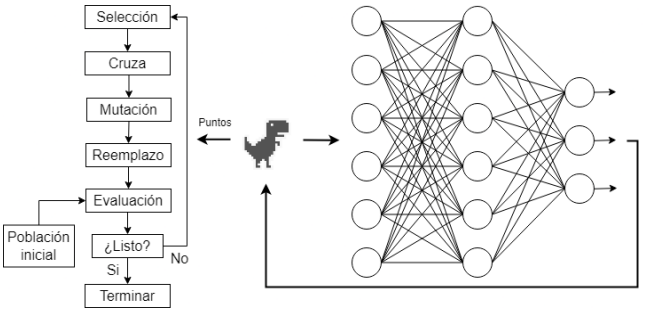

Fig. 3: Diagrama de solución con algoritmo evolutivo

permita identificar a los individuos más aptos para resolver el problema, es decir, más aptos para evitar obstáculos del juego. Al ir evolucionando y aprendiendo cómo evitar obstáculos, el algoritmo evolutivo irá modificando los pesos utilizados en la red neuronal mediante cruzas y mutaciones para ir generando individuos cada vez más aptos. En la Fig. 3 se muestra un diagrama general con la resolución antes mencionada.

Para realizar el proceso evolutivo se comienza con una población de individuos inicializada al azar, colocando aleatoriamente los pesos de la red neuronal para cada individuo, y se implementaron operadores de selección como ventana, competencia y ruleta, operadores de variación o reproducción para realizar cruzas y mutaciones, elitismo para conservar el mejor individuo de cada generación y el reemplazo población combinando los padres seleccionados con los hijos de la nueva generación.

Respecto a la función de fitness, aprovechando que al avanzar en el juego aumenta la puntuación obtenida, se utiliza dicha puntuación como valor de fitness para cada individuo, pudiendo así reconocer cuál es el individuo que mejor resuelve el problema.

Los operadores de selección utilizan el fitness obtenido por cada individuo para seleccionar los progenitores de la próxima generación. Cada método de selección permite tener una probabilidad mayor de ser elegidos para aquellos individuos más aptos para resolver el problema, sin perder la diversidad de la población dando la posibilidad de ser elegidos también a los menos aptos.

Luego de seleccionar los individuos padres, se utilizan operadores de variación para generar a los hijos de la próxima generación. En primer lugar el operador de cruza se utiliza con una probabilidad alta, entre 80% y 90%, y en caso de aplicarse la cruza, de los padres seleccionados se elige de manera aleatoria las neuronas de una misma capa y se las intercambian. En caso de que no se aplique la cruza, los individuos hijos serán una copia de los progenitores elegidos. Por otra parte se tiene el operador de mutación, donde se ha utilizado una probabilidad de mutación a nivel de individuo de entre 10% y 15%. En los casos donde se aplica la mutación, se elige de manera aleatoria los pesos sinápticos de una neurona y se les suma un valor generado también de manera aleatoria. Dicho valor se encuentra acotado según el desempeño del individuo que mejor juega durante toda la generación, utilizando la ecuación $\pm\,(0.1 + e^{-\alpha x})$, donde $\alpha = 0.001$. En la Fig. 4 se muestra dicha cota de mutación, donde en el eje horizontal se muestra la puntuación máxima de la población y en el eje vertical la cota para la mutación. Al utilizar esta cota, cuando el desempeño de la población es malo se permite

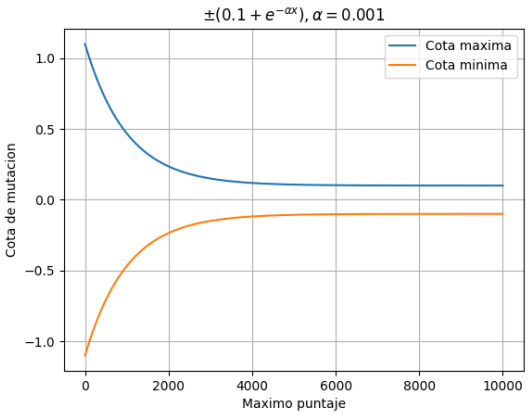

Fig. 4: Cota superior e inferior del valor a sumar en la mutación

realizar mutaciones más grandes a los individuos, ya que los valores que se sumarán serán altos, mientras que en el caso contrario el valor a sumar será chico y de esta manera se realizan variaciones pequeñas para mantener una convergencia razonable.

Luego de aplicar los operadores de variación y generar la cantidad de hijos necesaria, se compone la nueva población combinando los padres seleccionados y los hijos obtenidos. Además se utiliza elitismo para permitir que el mejor individuo de una población pase a la siguiente población sin modificaciones para mantener esa mejor solución obtenida hasta el momento.

III. PRUEBAS Y RESULTADOS

Tanto para el entrenamiento como para la validación, se crearon archivos con extensión .csv en los cuales se añadieron patrones con los datos de entrada a la red neuronal, obtenidos al jugar una persona y registrar cada vez que se presionaba una tecla, ya sea para saltar, agacharse o seguir corriendo. En el caso del dataset para entrenamiento, se registraron 600 patrones para cada tipo de acción que puede realizar el dinosaurio, mientras que para la validación se registraron 100 patrones para cada tipo de acción, obteniendo en total 1800 patrones para entrenamiento y 300 patrones para validación, teniendo así ambos datasets balanceados para los tres tipos de salidas de la red. Cabe aclarar que se utilizó una codificación one-hot para que solo una de las salidas de la red pueda tener como valor 1 y las dos restantes valor 0.
Tanto los datos de entrenamiento como validación fueron obtenidos jugando hasta lograr alrededor de 4.000 puntos en el juego.

Luego de realizar el entrenamiento, validación y obtener un resultado aceptable en el porcentaje de errores de clasificación, se almacenan en un archivo .csv los pesos obtenidos para la red neuronal. Al momento de querer utilizar la red neuronal para controlar el juego, se crea una red neuronal para el personaje, a la cual se le cargan los pesos obtenidos y almacenados anteriormente, y desde el juego se capturan los datos de entrada cada cierta cantidad de frames para ser enviados a la red y realizar solo la propagación hacia adelante para obtener como salida la acción que debe realizar el personaje, y de acuerdo a dicha salida simular si debe saltar, agacharse o seguir corriendo.

Al probar la implementación realizada con el MLP, se analizaron distintas arquitecturas de red y parámetros como tasa de error aceptable, número máximo de épocas y tasa de aprendizaje. Los parámetros utilizados durante el mejor entrenamiento logrado fueron:

-Arquitectura: [6, 6, 3]
-Tasa de error aceptable: 1%
-Número máximo de épocas: 300
-Velocidad de aprendizaje: 0.001
-Parámetro de la función sigmoidea: 1

Durante la etapa de validación del entrenamiento del MLP se obtuvo una tasa de error de clasificación individual para cada categoría y una tasa de error general, como se muestra en la Fig. 5, donde en el eje horizontal se muestra la cantidad de épocas y en el eje vertical la tasa de error de clasificación, siendo el color rojo la categoría saltar, el azul la categoría agacharse y el verde la categoría correr, mientras que la curva en color negro representa la tasa de error general. Por otro lado, en la Fig. 6 se muestra la curva de error cuadrático medio durante la validación, donde en el eje horizontal se muestra la cantidad de épocas y en el eje vertical el error cuadrático medio. Se puede observar como el error cuadrático medio disminuye al aumentar la cantidad de épocas, lo cual nos permite saber que el modelo está aprendiendo durante el entrenamiento.

Al realizar diversas instancias de entrenamiento de la red neuronal se ha comprobado que la convergencia del modelo puede variar mucho de acuerdo a la inicialización aleatoria de los pesos. Con el mejor entrenamiento realizado se obtuvo una tasa de error del 1.3%, lo que es igual a un 98.7% de aciertos, y si bien se había utilizado 300 como número máximo de épocas, en la Fig. 5 se puede observar que se llegó a dicho resultado alrededor de 50 épocas y luego sólo hubo pequeñas variaciones. Utilizando los pesos obtenidos en dicho entrenamiento, con el MLP se logra alrededor de 11.000 puntos en el juego.

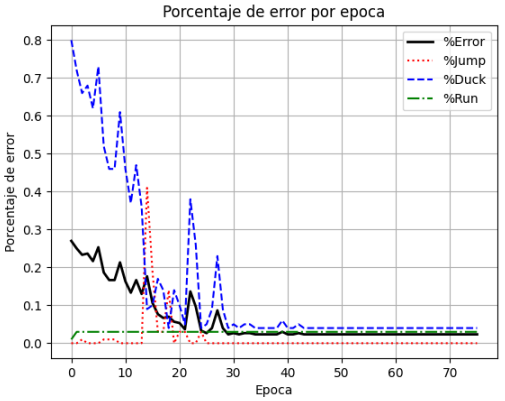

Fig. 5: Error de clasificación general y por categoría

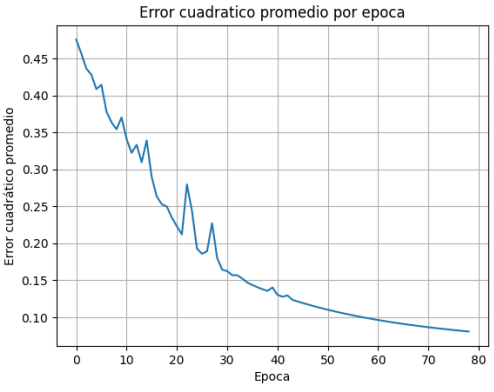

Fig. 6: Error cuadrático medio durante la validación

Por el lado del algoritmo evolutivo, pensando en poder comparar su funcionamiento contra el MLP, para comenzar se han realizado una serie de ejecuciones y se determinó que los parámetros a utilizar en el algoritmo serían:

-Arquitectura de red [6, 6, 3]
-Parámetro de la función sigmoidea: 1
-Cantidad de padres: 50%.
-Probabilidad de cruza: 90%
-Probabilidad de mutación a nivel de individuo: 15%

Para determinar cuál es el mejor método de selección para utilizar se decidió realizar 10 generaciones con cada método y con la población final obtenida se realizaron 10 ejecuciones más pero desactivando las cruzas y mutaciones para que se mantenga la misma población. A partir de estas 10 ejecuciones se registraron las puntuaciones máximas obtenidas al jugar y se calculó una media y desvío para cada método, obteniendo los resultados que se muestran en la Tabla I, donde en las columnas se observa el método de selección utilizado y en las filas la media y desvío de la puntuación o valor de fitness obtenido, con lo cual se pudo determinar que el mejor método de selección para este caso es el de competencia. Se comprobó que estos valores pueden aumentar si se realiza una mayor cantidad de generaciones, sin embargo, para realizar esta prueba se determinó utilizar ese número de generaciones.

TABLA I
MEDIA Y DESVÍO DE FITNESS EN MÉTODOS DE SELECCIÓN

	Ventanas	Competencia	Ruleta
Media	12.936,1	13.908,2	6.017,5
Desvío	374,76	441,71	557,52

Otra de las pruebas realizadas fue entrenar el algoritmo evolutivo durante 10 generaciones pero con tres arquitecturas distintas para la red neuronal. En la Fig. 7 se observan los resultados obtenidos, donde en el eje horizontal se muestra el número de generaciones realizadas y en el eje vertical el fitness o puntuación alcanzada. En azul se grafica la arquitectura [6, 6, 3], en naranja [6, 3, 3] y en verde [6, 8, 3]. Se puede comprobar que la arquitectura [6, 6, 3] que fue la seleccionada arroja un resultado notablemente mayor a las demás.

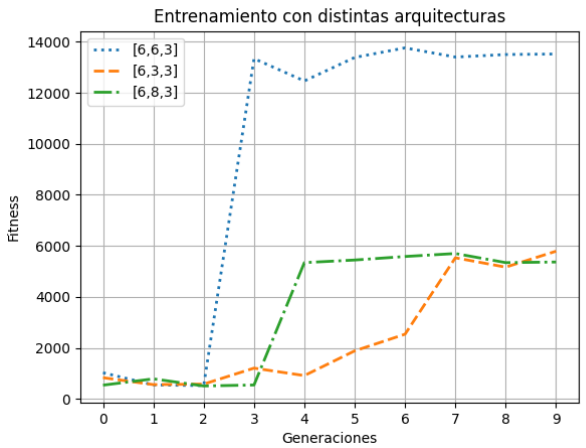

Fig. 7: Entrenamiento de algoritmo evolutivo con distintas arquitecturas de red neuronal

Las dos soluciones implementadas han logrado un resultado muy satisfactorio al aprender a jugar, ya que utilizando el MLP entrenado con datos sólo hasta alrededor de 4.000 puntos en el juego, se logró una media de 11.743 puntos, y con el algoritmo evolutivo luego de 10 generaciones se obtuvo a una media de 13.908 puntos con el mejor método obtenido. Cabe destacar que se podría aumentar aún más la puntuación obtenida por el MLP si se entrenara con datos de mayor puntuación, y un mejor resultado en el algoritmo evolutivo si se realizan más generaciones de entrenamiento.

IV. CONCLUSIONES

Por una parte se pudo comprobar que se logra una efectividad mucho mayor utilizando las soluciones implementadas con inteligencia artificial en comparación con el jugador humano. Por otro lado, comparando las dos soluciones implementadas, se llegó a la conclusión de que con el mejor modelo de algoritmo evolutivo se obtiene un mejor resultado pero con la desventaja de necesitar mayor tiempo para obtener generaciones aptas para resolver el problema, mientras que el MLP tiene la ventaja de que en poco tiempo de entrenamiento también logra un excelente resultado.
Otros detalles importantes a destacar es que el MLP únicamente aprende a partir de los datos de entrenamiento que le fueron dados, es decir, aprenderá de la forma de juego del humano que registró esos datos, mientras que el algoritmo evolutivo se ha observado que por su propia cuenta aprende a realizar acciones interesantes como combinaciones de saltar un obstáculo y bajar rápido para ganar tiempo en la próxima decisión. Otras veces se ha observado que algunos individuos encuentran como forma de juego correr agachados todo el tiempo y solo saltar, o que evitan saltando algunos obstáculos que naturalmente se los evitaría agachándose, pero que igualmente es una opción válida de juego.

V. REFERENCIAS

[1] Ding, S., Li, H., Su, C. *et al.* “Evolutionary artificial neural networks: a review”. *Artif Intell Rev* 39, 251–260 (2013). https://doi.org/10.1007/s10462-011-9270-6

[2] Marwah, Divyanshu, et. al. “Chrome Dino Run using Reinforcement Learning”, ArXiv preprint arXiv:2008.06799, 2020.

[3] Bullen, T., and M. Katchabaw, “Using genetic algorithms to evolve character behaviours in modern video games”, *Proceedings of the GAMEON-NA*, 2008.

[4] Simpson, Reed. “Evolutionary Artificial Intelligence in Video Games”. *University of Minnesota*, 2012.

[5] K. Chellapilla and D. B. Fogel, "Evolution, neural networks, games, and intelligence," in Proceedings of the IEEE, vol. 87, no. 9, pp. 1471-1496, Sept. 1999, doi: 10.1109/5.784222.